

PASSIVE RANGING OF NEAR FIELD TARGETS IN THE AUDIO FREQUENCY BAND

Ian Robert Howard Dennis

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



1997

Full metadata for this item is available in
St Andrews Research Repository
at:

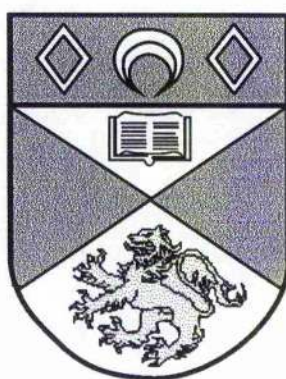
<http://research-repository.st-andrews.ac.uk/>

Please use this identifier to cite or link to this item:

<http://hdl.handle.net/10023/13610>

This item is protected by original copyright

Passive Ranging of Near Field Targets in the Audio Frequency Band



A thesis presented by
Ian Robert Howard Dennis
to the
University of St. Andrews
in application for the degree of
Doctor of Philosophy

August 1997



ProQuest Number: 10167196

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10167196

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Th c 372

Declaration

I, Ian Dennis, hereby certify that this thesis, which is approximately 31000 words in length, has been written by me, that it is a record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

I was admitted as a research student in October 1994 and as a candidate for the degree of Ph.D. in October 1995; the higher study for which this is a record was carried out in the University of St. Andrews between 1994 and 1997.

In submitting this thesis to the University of St. Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any *bona fide* library or research worker.

Ian R.H. Dennis

August 1997

Certificate

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of Ph.D. in the University of St. Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Jim C.G. Lesurf

August 1997

Acknowledgements

I would like to thank Jim Lesurf for all of his patience, help and guidance throughout the course of my Ph.D. He also deserves thanks for his good humour and friendship which have often given me inspiration at times when I needed it most.

Appreciation also goes to the members of the Millimetre Wave Group. In particular thanks go to Duncan Robertson and Graham Smith who gave an endless supply of help when I first started this Ph.D.

A large amount of guidance in programming was also given by Andy Ray of Intelligent Interfaces. He deserves credit for showing me the correct way to structure programs which resulted in making the software routines in this thesis so powerful.

I am also grateful to the Engineering and Physical Sciences Research Council and to the Defence Research Agency for financial assistance. Without this funding it would not have been possible to undertake this Ph.D.

Finally, thanks goes to my family and friends who have made my time in St. Andrews very rewarding.

Abstract

The use of passive ranging techniques for imaging targets situated in the near field for the audio frequency band is presented in this thesis.

A three port interferometer used in millimetre waves is introduced and it is shown that with some approximations, it can be used to passively image targets in the medium-far field. The geometry of this system is then adapted for use in the near field in the audio frequency band, without the need for making any approximations. A model is then constructed to demonstrate the theory. The design, construction and testing of the new system for use in an anechoic chamber is then presented.

This thesis also introduces a number of digital signal processing techniques that take full advantage of the recent increase in computing power and decrease in costs. These techniques are then used to perform frequency analysis on real data obtained experimentally from the new three port system.

A number of different experiments and results are then presented. A loudspeaker is used as a test source and it is shown that the two separate drive units can be distinguished very accurately with the system. The whole system is then fully characterized and the optimal system setup conditions are found.

Finally some ideas for future work and possible applications are presented. It is shown that the system would make an ideal tool for testing the spatial coherence properties of loudspeakers.

Table of Contents

Declaration	<i>i</i>
Certificate	<i>i</i>
Acknowledgement	<i>ii</i>
Abstract	<i>iii</i>
Table of Contents	<i>iv</i>
Chapter One	
Thesis Overview	1
1.0 Introduction	1
Chapter Two	
Introduction and Techniques	4
2.0 Introduction	4
2.1 Propagation of Sound in Air	4
2.2 Microphones as Receivers	5
2.3 Loudspeakers as a Noise source	7
2.3.1 Drive Units	8
2.3.2 Baffles	9
2.3.3 Frequency Response	11
2.4 White Noise	11
2.5 Introduction to Frequency Analysis	14
2.6 Early Experiments	17
2.6.1 Frequency Response Using the Stanford Spectrum Analyser	18
2.6.2 A Computerised Data Acquisition System	20
REFERENCES	21
Chapter Three	
Three Port Geometry and Modelling	22
3.0 Introduction	22
3.1 Introduction to Ranging	22
3.2 A Three Port Interferometer	27
3.3 An Improved Three Port Interferometer	31
3.3.1 Linear Port Spacings	34
3.4 Model For Data Generation	35
3.5 System Characterization and Errors	43
3.5.1 Different Port Spacings	44
3.5.2 Different x Co-ordinates for the Target	45
3.5.3 Different z Co-ordinates for the Target	46
REFERENCES	48

Chapter Four

4.0 Introduction	49
4.1 New System Requirements	49
4.2 New System Overview	51
4.3 White Noise Sources and Other Test Signals	53
4.3.1 Compact Disc Test Signals	53
4.3.2 Pseudo Random White Noise	53
4.3.3 Transistor White Noise Source	54
4.3.4 HP Arbitrary Waveform Generator	56
4.4 Microphones and Microphone Preamplifiers	56
4.4.1 Large Microphones	56
4.4.2 Small Microphones	56
4.4.3 Microphone Frequency Response Comparisons	57
4.4.4 Microphone Preamplifiers	59
4.5 Control Box Circuitry	60
4.6 Data Acquisition System	61
4.7 Data Processing	63
4.8 Anechoic Chamber Setup	64
4.9 System Testing	65
REFERENCES	66

Chapter Five

Digital Signal Processing	67
5.0 Introduction	67
5.1 Hardware and Software Issues	67
5.1.1 Platform Choice	67
5.1.2 Programming Language Choice	69
5.1.3 Memory Allocation	70
5.1.4 General Software Issues	71
5.2 File Library (Filz)	71
5.3 Digital Signal Processing Library	73
5.3.1 Fast Fourier Transform (FFT) Algorithm	74
5.3.2 Correlation Using a FFT	78
5.4 DSP Utility Library	80
5.4.1 DC Offset Removal (rmdc)	81
5.4.2 Apodising Function (Bartlett)	81
5.4.3 Phase Function (phase)	81
5.4.4 Power Function (power)	82
5.4.5 Power Averaging Function (smoother)	82
5.4.6 Decibel Scale Function	83
5.4.6.1 Zero dB Calibration	83
5.4.7 Phase Correction Function (PhaseCrct)	84

5.4.8 Phase Differential Function (diff)	85
5.4.9 Constrained Regression Function (ConReg)	85
5.5 System Utility Library	86
5.6 Other Libraries	87
5.6.1 Statistics Library (stats)	87
5.6.2 Range Library (Range)	88
5.7 Testing of Library Functions and Routines	88
5.8 An Example: A Four Channel Analyser	89
5.9 Conclusion and Comments	90
REFERENCES	91

Chapter Six

Experiments and Results	92
6.0 Introduction	92
6.1 First Three Port Experiment	93
6.2 Three Port Experiment With The Loudspeaker Sideways	98
6.3 Expansion to Five Ports	99
6.4 Five Ports With Variable Port Spacing	100
6.4.1 20cm Port Spacings	100
6.4.2 30cm Port Spacings	102
6.4.3 40cm Port Spacings	104
6.4.4 General Comments	106
6.5 Five Ports With Variable Target Co-ordinates	107
6.6 Five Ports With Sideways Orientated Source	110
6.7 Comments and Conclusions	113

Chapter Seven

A Final Review	115
7.0 Conclusion	115
7.1 Future Work	117
7.2 Possible Applications	118

Appendix A	120
-------------------	-----

Appendix B	126
-------------------	-----

CHAPTER 1

Thesis Overview

1.0 Introduction

It is perhaps a little unusual to include an overview at the beginning of this thesis, but nevertheless it is something which was considered very necessary by the writer. The work which is presented in the following chapters does not by any means give any form of proportional representation to the time spend on different subjects. This was partly due to the nature of the work that is presented and partly due to circumstances that were imposed at the beginning of the Ph.D. study. It is for these reasons that a short overview of the amounts of time spent on different areas is presented here.

In the first few months one of the main objectives was to come to terms with as many new of the new concepts as possible that were required to fully understand ranging and signal processing techniques. Quite a large amount of this work has been presented in the introduction and techniques in chapter 2. At the same time an anechoic chamber had been recently acquired, and contained mostly old an obsolete equipment. It was quickly realised that there was hardly any equipment that was going to be suitable for passive ranging experiments and so a lot of new equipment was going to be required. It was decided that most of this equipment should be hand built, although it was recognised that this would be time consuming. There were two main reasons for this. First, there was a lack of money for purchasing new equipment, which would have been very expensive. Second, there was some equipment that could not be purchased with the correct specification - an example of this would be the pseudo random noise generator that is described in chapter 2 & 4. In both cases the equipment was built relatively cheaply and performed reasonably well.

After enough equipment had been built, some initial experiments were carried out in the anechoic chamber. One such experiment used a pseudo random white noise generator to characterize the frequency response of a loudspeaker - this is detailed in chapter 2 in section 2.6.1. Once that it had been established that the loudspeakers frequency response had been characterized successfully, the next obvious move was to start building a computerised data acquisition system. It was at this time that the writer invested a lot of time learning the C/C++ programming language, so that digital signal processing routines could be written on a computer. The computerised data acquisition system was based around a "Wild Vision Analogue to Digital" interface board and an Acorn Archimedes computer. Unfortunately the construction of this system was unsuccessful due to several unforeseen circumstances. The construction and problems that arose are covered fully in chapter 2 in section 2.6.2. After the system had first been tested, six months were wasted trying to get it to work properly, before it was realised that a new approach and a new interface board would be needed. By now almost a year and a quarter had elapsed.

It was at this stage when the writer was fortunate enough to be awarded a "CASE" studentship from the Defence Research Agency (DRA). This meant that more money was available to purchase new equipment at a time when it was needed the most. The writer also assisted the DRA in performing some scaled sonar experiments in the anechoic chamber. This work was not of direct relevance to this thesis and accordingly has not been included, although the data acquisition board which was used for the experiments was kindly lent afterwards for use in the anechoic chamber. This was a very welcome gesture and helped recover some of the time lost trying to get the previous system to work properly. This new data acquisition board cost approximately £5000 and fortunately had the ideal specifications for use in passive ranging experiments. The only bad point was that it was only

compatible with an IBM PC computer whilst all of the previous work and C programming had been carried out on the Acorn platform. This is discussed in more detail in chapter 4 and chapter 5.

From this point on, the rest of the time was spent writing software, designing, building, and testing the passive ranging system that is presented in the following chapters. A brief introduction is given to the basic components and factors needed for passive ranging in chapter 2. In chapter 3, an introduction to ranging is presented along with three port geometry and a model. In chapter 4, the design and construction details for all of the components that make up the ranging system are presented. All of these components were built specially for use with the new data acquisition board. The digital signal processing routines that were written in the C programming language are fully explained and documented in chapter 5. In chapter 6, results taken using the completed system are presented, and used to fully characterise the system. Finally future work and applications are discussed in chapter 7.

CHAPTER 2

Introduction and Techniques

2.0 Introduction

In this chapter an introduction is given to the conditions and some of the basic components that are used in three port passive ranging systems. In section 2.1 consideration is given to the propagation of sound in air. Microphones and loudspeakers are then presented because they formed the basic means of receiving and transmitting signals in experiments. Design and construction theory for pseudo random white noise sources is then discussed. An introduction to frequency analysis is then presented to give a brief introduction and provide some background to the digital signal processing in chapter 5. Finally some early experiments and findings have been included that demonstrate how some of the theory presented in this chapter can be used in experiments. These early experiments were particularly significant because the results provided new a new direction for later work. The introduction to passive ranging and three port systems is not covered here and can be found in chapter 3.

2.1 Propagation of sound in Air

Sound is generally defined as any pressure variation in air, water or any other medium that the human ear can detect. The number of pressure variations per second is called the frequency of the sound and is measured in hertz. Typical hearing ranges for normal healthy people ranges from 20Hz to 20kHz. This is quite a large range considering that the range from the lowest note to the highest of a piano is 27.5Hz to 4186Hz. These figures represent the ideal human being, but in practice as we become older, we become less sensitive to high frequencies and an upper frequency limit of around 16kHz is

by no means unusual. Sound waves in air are longitudinal waves propagated through the medium by the transfer of kinetic energy from one molecule to another. The restoring force for such a wave is supplied by the pressure of air. As a sound wave spreads out from its source, the intensity falls off because the area of the wave front grows larger and hence the wave energy per unit area becomes smaller. The total power carried by the wave front remains the same, and as the radius grows the intensity or power per unit area varies as the inverse square of the distance. The velocity of sound in air can be calculated using the following formula,

$$c = \sqrt{1.40 \frac{P_0}{\rho_0}} \quad 2.1.1$$

where P_0 represents the unperturbed gas pressure (N/m²) and ρ_0 is the the gas density (kg/m³). The factor of 1.40 is the ratio of the principal specific heat capacities of the gas. The speed of sound in air is generally accepted to be 331 m/s at 0°C and 344 m/s at 20°C. Finally the wavelength is related to the frequency by the following expression,

$$c = f\lambda \quad 2.1.2$$

where c is the speed of sound (m/s), f is the frequency (Hz), and λ is the wavelength (m). Further information on the propagation of sound in air may be found in either Blitz¹, Ford² or Brown³.

2.2 Microphones as receivers.

In this section consideration is given to the properties of microphones in general, and in particular to the unidirectional cardioid type that were used experimentally in this thesis. Performance of various sizes of

microphone is looked at in terms of cost and build quality, and compared with theoretical performance. This is quite an important issue, because many microphones were needed for the experiments carried out in this study, which meant that relatively low cost was a requirement.

A microphone can be described as a transducer that converts acoustic energy into electrical energy. Microphones are used to convert the periodic variations of acoustic pressure in air into similar variations in voltage or current in an associated electrical circuit. There are two basic types of microphone. The classification of pressure microphone is given to microphones that have an electrical response that corresponds to the variations in acoustic pressure. The other type is known as a pressure gradient microphone, where the electrical response corresponds to variations in the pressure gradient. An example of a pressure gradient microphone is what is commonly known as a velocity microphone. The microphones that were used as receivers in the experiments in this thesis were of the unidirectional cardioid type. This type of microphone combines a pressure element and a velocity element in series. The directional response characteristics of these two elements can be seen below in figure 2.1 in response (1) and (2).

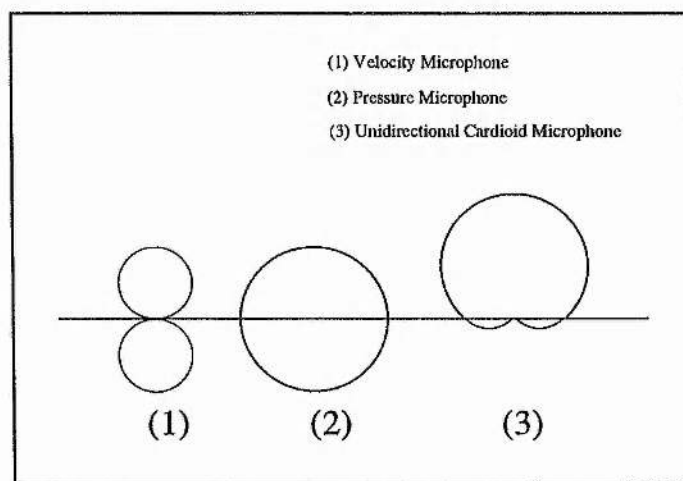


Figure 2.1: Directional response characteristics of various microphones

The other response (3) is obtained from the combination of the first two responses and gives a characteristic that is a cardioid of revolution. The result is that unidirectional cardioid microphones receive sound over a large solid angle without appreciable frequency discrimination. This was the main reason that this type of microphone was used in experiments. The other areas that need to be addressed concerning microphones are cost, frequency response, and size. In theory, the smaller the microphone, the better the frequency response. This means that there is less averaging and less spatial coherence problems, although there is less sensitivity compared with a larger microphone. There is also the problem that it is quite difficult, and hence very expensive, to build good quality small microphones. Generally for low to mid price range, larger microphones usually offer better performance and frequency response than small ones. It is only in the high price range that smaller microphones may out perform large microphones. More information on microphones can be found in Beranek⁴ or Kinsler & Frey⁵

2.3 Loudspeakers as a noise source

This section considers the properties of loudspeakers as a noise source. A brief introduction is given to show why loudspeakers do not behave as point sources and why they have imperfect frequency responses. A loudspeaker can be described as a transducer operating from an electrical system to an acoustical system and designed to radiate sound. Typically a loudspeaker consists of one or more driving units, linked together by a crossover network, which are mounted in some form of baffle or housing. First we shall consider the driving units and look at why more than one is usually required. Baffles and enclosures will then be discussed and finally consideration will be given to how all of these items effect loudspeaker frequency response.

2.3.1 Drive Units

Here a brief introduction is given to the properties of typical drive units that are used in loudspeakers. This section makes no attempt to cover all the different properties, construction techniques and materials that could be employed - there are many good texts on this subject such as that by Sinclair & Newnes⁶ that can be consulted if required. A cross section of a typical drive unit can be seen below in figure 2.2. It is the movement of the cone which produces the sound waves and this is the item to which we will give consideration. The cone is usually made out of paper or a polymer, although other materials may also be used, and is typically fixed to the outer edge of the frame by a flexible roll of cloth or rubber. The cone shape is used because of its strength. There are corrugations where the cone is fixed directly to the frame to permit forwards and backwards movements whilst preventing any sideways movement. Every physical object will have a fundamental resonance at a certain frequency and this includes loudspeaker cones. This leads to an uneven frequency response because the output at resonant frequency is greater than at all others.

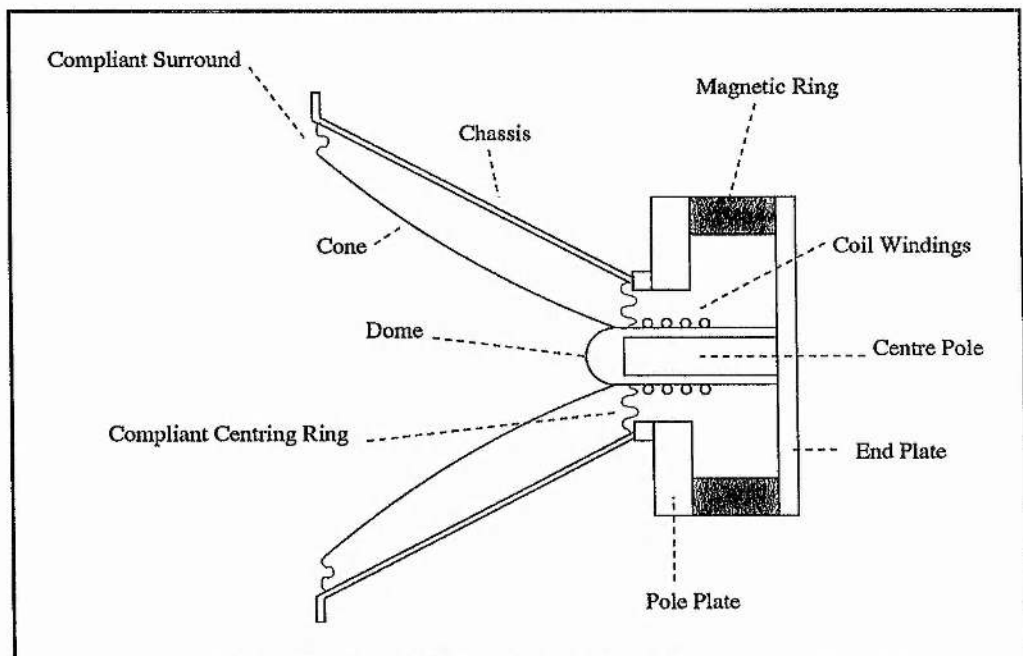


Figure 2.2: Cross section of a loudspeaker drive unit

Below the resonant frequency, the output falls off at a rate of about 10 dB per octave, which means to achieve a good bass response the resonant frequency should be as low as possible. The formula for the resonant frequency can be seen below:

$$f_r = \frac{1}{2\pi\sqrt{MC}} \quad 2.3.1$$

where M is the mass of the cone and C is the compliance of the suspension (the inverse of the suspension stiffness). This means that a high mass and compliance are required for a low resonant frequency. For an extended low frequency response, apart from the cone needing to have a large mass, the diameter of the cone also needs to be large because the efficiency of the cone falls off with decreasing diameter at low frequencies. This is in direct conflict with the requirements for a good high frequency response, where the cone should be both small and light. The conflict in requirements means that generally two separate drivers are used for the treble and bass and these are often referred to as tweeters and woofers respectively. Tweeters work using the same basic principles as woofers (we have described a typical woofer up to this point), although there are a few small differences. The back of the unit is usually fully enclosed to prevent nearby woofers causing interference and often the cone is reversed to form a dome to give a wider angle of dispersion. By using two driver units it is necessary to use a cross over network to send the correct frequencies to the appropriate drive unit. This adds further imperfections and complexity to the overall system.

2.3.2 Baffles

When a signal is played through a driver in free space two separate sound waves are produced, one coming from the front and the other coming 180 degrees out of phase from the rear. This means that the two signals may interfere and we get cancellation at the rim of the driver unit. The driver then

behaves as a high frequency source for wavelengths where the wavelength is smaller than the cone diameter. This is the reason that driver units sound tinny when sound is played through them when they are not mounted in a baffle.

The purpose of using a baffle is to keep the two waves that a driver produces apart. By using a flat baffle, the two waves still meet at the edge of the baffle but now have further to go. This means that larger wavelengths can be propagated before cancellation starts occurring because the overall diameter is larger - i.e. the bass response is extended. One advantage in using a plain baffle is that there is a lack of air resonance that is associated with enclosures and that produces colouration. Panel resonance are also a minimal. One disadvantage with flat baffles is that because the waves emitted from the rear of the baffle take a finite time to reach the edge, a wave from the rear can occur at the same time as the next wave from the front. Reinforcement occurs at wavelengths that are 0.5, 1.5, 2.5, etc. times the radius of the baffle. Conversely cancellation occurs at wavelengths of whole multiples of the baffle radius. The result of the interference is that an uneven frequency response with lots of peaks and troughs is obtained. One solution is to use a rectangular baffle and to mount the driver off centre, thus making all of the distances to the edge of the baffle in each direction different. This can be quite difficult to achieve when more than one driver units are needed. There is however another problem that make flat baffles unusable in practice. An open baffle would need to have a radius of approximately 8 metres to give a flat frequency response down to around 45 hertz. Clearly this is not a practical option for domestic use. Even if a 0.5 metre radius (which is barely practical) was used, frequency roll off would occur at around 300 hertz. Whilst this may be acceptable for speech, it certainly would not be suitable for music and would be detrimental to the clarity.

The problems associated with open baffles can be overcome to some extent by using what is commonly known as an infinite baffle. In effect an

infinite baffle can be made by folding a baffle into a closed box shape. By using an infinite baffle the frequency range is extended enough to produce adequate bass, however more resonance is obtained. The enclosure produces resonance that is dependent upon its size and the extra panels that are used generate panel resonance. Both of these resonance can be minimised to some extent but are detrimental to the overall frequency response. There is a lot of work that has been done on this subject and it would be inappropriate to include here as it is not directly relevant to this thesis, however more details, if required, can be found in Morse⁷ or Capel⁸.

2.3.3 Frequency Response

As we have seen in the previous sub-sections, loudspeakers are far from ideal sound sources. The driver units do not have flat frequency responses and it is usually necessary to use two drivers, namely a woofer and a tweeter. When two drivers are used, an appropriate cross over network is required, which filters the input signals. Also, by using two drivers additional spatial coherence problems arise. Once we place driver units in an infinite baffle the frequency response suffers further due to enclosure and panel resonance. Coupling all of these things together, it should come as no surprise that a typical good quality "high fidelity" loudspeaker, has a frequency response that is only flat to within about 5 decibels. This fact also means that white noise sources that are flat to within 0.1 decibels are often used to obtain loudspeaker frequency responses, because the fluctuations in the noise source are insignificant compared to those of the loudspeaker under test. A typical frequency response of a loudspeaker can be seen in figure 2.6 in section 2.6.1.

2.4 White Noise

White noise is a noise that can be defined as consisting of a mixture of harmonic waves of all frequencies with equal intensities. It is a useful test

source because it has flat power spectrum up to a defined cutoff frequency, making it ideal for measuring frequency responses of devices that have an uneven frequency response. White noise sources come in a variety of forms. Audio test compact discs often have high quality white noise source tracks, although in practice they are not that useful due to their short duration. White noise can also be found on a variety of modern function generators. Typical generators of this type use a Read Only Memory, in to which various source types have been preprogrammed, to send digital signals to a digital to analogue converter. Since Read Only Memories are of a finite size, this inevitably means that at a certain time the signal will repeat itself. If this period is too short, periodic effects will be seen at low frequencies when data is sampled for a long time using a high sampling rate. White noise can also be generated using analogue techniques and more details about this subject can be found in chapter 4. Another way of creating white noise is to use an interesting blend of analogue and digital techniques. It is this type of white noise source that was mostly used for the experiments in this study, and accordingly it is given the most attention in this section. First we will consider the digital part of such a generator. Pseudo random bit sequences can be generated quite simply by using a few shift registers and a little logic - see figure 2.3 below.

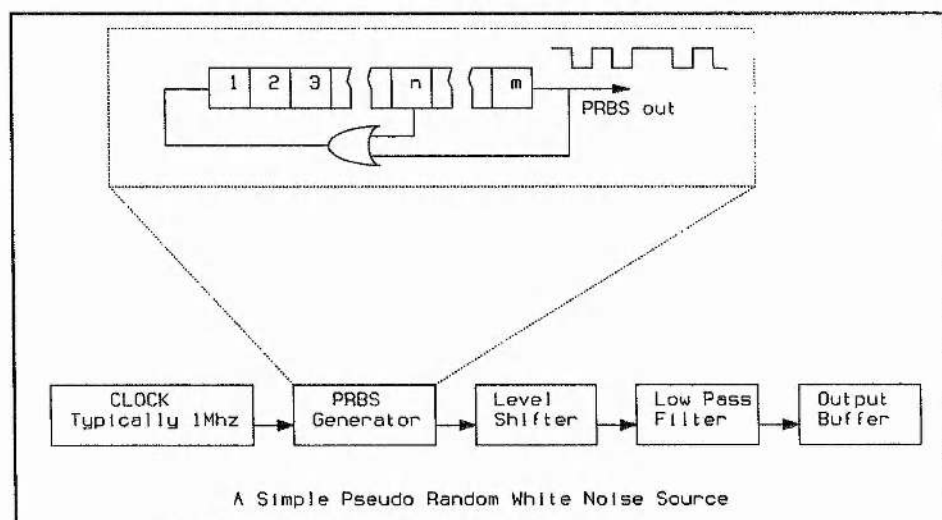


Figure 2.3: Schematic for a pseudo random white noise source

A series of shift registers are linked together and a digital sequence is formed by using an exclusive OR gate to create a feedback loop. A clock is then used to drive the shift registers and a pseudo random sequence can then be observed from the output. It is termed a "pseudo random" sequence because it appears as if it is random, although in reality the sequence can be predicted if the number of shift registers and the location of the feedback taps is known. By passing the pseudo random sequence (PRBS) through a low pass filter, band limited white Gaussian noise is produced with a flat power spectra. The useful part of the noise spectra that we are interested in ranges from the low frequency PRBS repeat period, up to an upper frequency of about 12% of the clock frequency used to drive the PRBS sequence. Within these limits the noise is flat to $\pm 0.1\text{dB}$. The feedback taps leading to the exclusive-OR gate are chosen such that a maximum length sequence is obtained. If we have m bits in our shift register sequence, the maximal length sequence will be $2^m - 1$, assuming m and n are chosen correctly. To achieve this, reference 9 tells us that the polynomial $1 + x^n + x^m$ should be irreducible and prime over the Galois field. The one state we want to avoid is when all the bits in the shift register are at zero which will lead to a steady state. The anechoic chamber was already equipped with a white noise box that was built used a shift register sequence of 17 bits with feedback taps of $m=17$ and $n=14$. This produced a sequence length of 131071 and as the sequence was clocked at 1MHz a reciprocal period of 0.13 seconds was obtained. As we will see in section 2.6.1 this repeat period was not ideal and caused structure to be seen at low frequencies. The good news is that the reciprocal period is relatively easy to change because it is basically dependent upon the length of the shift register and the speed of the clock that is driving it. In theory if a shift register with a length of 100 bits was driven by a 10Mhz clock, a repeat period a million times longer than the age of the universe could be obtained! More details of construction and theory of white noise sources can be found in Horowitz & Hill⁹ or Dixon¹⁰.

2.5 Introduction to Frequency Analysis

Over the last few years there has been a big swing from analogue to digital analysis techniques in almost all areas of frequency analysis. The main reason for this development is has been due to an increase in computing power coupled with a reduction in cost. As a result a number of dedicated spectrum analysers using FFT (Fast Fourier Transform) algorithms can now be purchased relatively cheaply. The object of frequency analysis is to break down a complex signal into its various frequency components. In this case when we use the word "components" it is intended to represent the results of a Fourier Analysis. The mathematical basis for this type of analysis is the Fourier Transform. The transform assumes that the signal is periodic and is composed of a number of sinusoidal components at various different frequencies, each having an initial starting phase and amplitude. A sinusoidal wave can be represented as the vector sum of two vectors with the same amplitude that rotate in opposite directions. This is shown in figure 2.4 below.

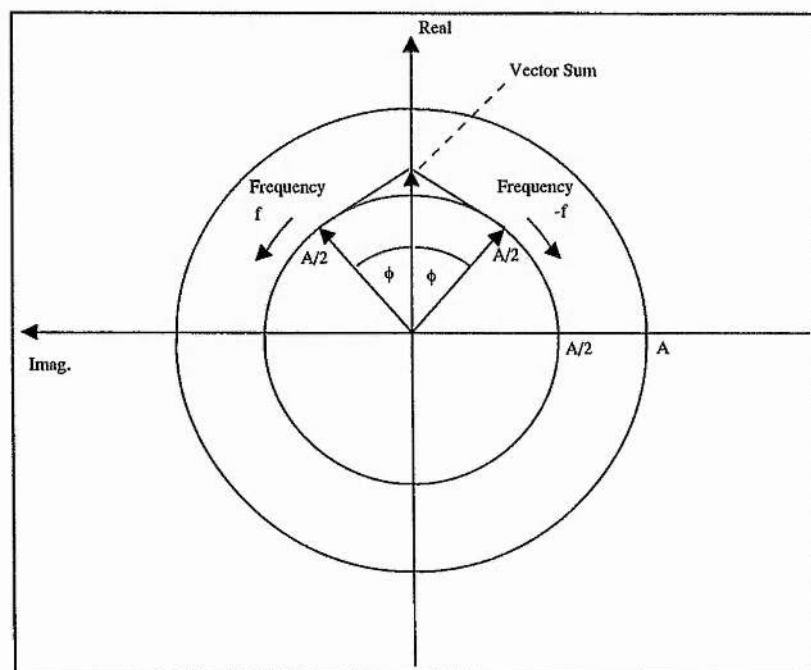


Figure 2.4: Contra-rotating vector representation of a sinusoidal component of the form $A \cos (2\pi ft + \phi)$

One vector of amplitude $A/2$ has an initial phase of ϕ and rotates with frequency f , while the other vector has an amplitude of $A/2$, a starting phase of $-\phi$ and rotates with frequency $-f$. The use of negative frequency is necessary here to indicate the rotation in opposite directions by indicating a negative rate of change of phase angle. As the two vectors rotate with time, the sum will always be real because the imaginary parts will always cancel out. The wave can be written in the following form,

$$A \cos \theta = \frac{A}{2} (e^{j\theta} + e^{-j\theta}) \quad 2.5.1$$

where

$$\theta = (2\pi f t + \phi)$$

We will now show how this can be applied to Fourier analysis. Let us suppose that we have a function that is periodic in time, i.e.

$$h(t) = h(t + nT) \quad 2.5.2$$

where t is the time, T is the periodic time, and n is any integer. When such a periodic function is expanded using a Fourier series expansion, it can be represented as a sum of sinusoidal components (or rotating vectors) at equally spaced frequencies. In the experiments in this thesis, the data was sampled discretely in time by using analogue to digital converters. The form the Fourier Transform takes for sampled time functions is shown below,

$$H(f) = \sum_{n=-\infty}^{\infty} h(t_n) e^{-j2\pi f t_n} \quad 2.5.3$$

and the reverse transform is,

$$h(t_n) = \frac{1}{f_s} \sum_{-f_s/2}^{f_s/2} H(f) e^{j2\pi f t_n} df \quad 2.5.4$$

where f_s is the sampling rate and $t_n = n\Delta t$ which represents the time at the n th sample. A more detailed treatment of this can be found in Proakis & Manolakis¹¹. This subject is also covered in more depth in Chapter 5.

By using an appropriate algorithm such as as Fast Fourier Transform, discretely sampled time series data can be transformed into the frequency domain. If a simple sinusoidal wave was digitized and processed using a FFT algorithm, the result in the frequency domain would be a delta function with the spike situated at the frequency value of the sampled waveform. If we were to sample a broad band source such as white noise, we would expect to see an almost horizontal line in the frequency domain. The algorithms needed to transform time series data are covered in full detail in chapter 5. Care should be taken when sampling that data to make sure that the Nyquist criterion is met. To avoid aliasing we need to ensure that the sampling frequency is at least twice the value of the maximum frequency component in our sample. Practically this requirement usually means that some form of analogue filtering is required to limit the maximum frequency value that is input to a analogue to digital converter.

Another technique that can be very useful is correlation. Correlation can be carried out in both the time and frequency domain and is usually used to determine a time difference between two signals. First let us consider correlation in the time domain. Suppose we have two signals, a and b , that are discretely sampled in time. Consider equation 2.5.5 below.

$$\sum_{\Delta=0}^{\Delta=t} a(t) * b(t - \Delta) \quad 2.5.5$$

The correlation is performed by introducing a set of different time delays (Δ) into one of the signals, which here we will call signal b . Every time delayed version of signal b is then multiplied by a copy of signal a , and a summation is made from the results. After this has been carried out for all of the time delayed versions of signal b , a new data set is formed which contains results for different values of time. When this new data set is plotted, a peak value is produced at the time value by which signal a leads or lags signal b . We will now look at how this process can be performed in the frequency domain. Apart from the inconvenience of having to transfer the signals into the frequency domain, the process is far simpler using this method. To start with, a FFT is performed on the data sets of both signals. Once the data is in the frequency domain, all that is required is to multiply the elements of one data set by the complex conjugate of the other. The resultant data set is then returned to the time domain by using an inverse FFT algorithm. Now the same results are obtained as performing the correlation in the time domain, although the method is much quicker.

All of the techniques described here have been used heavily throughout the experimental work in this thesis. Their use and application will be discussed in far more detail in the following chapters. More general details about these subjects can be found in Randall¹².

2.6 Early experiments

In this section some of the early experiments that were carried out at the beginning of this study are presented. They present a good opportunity to show how the theory from the rest of the sub-sections in this chapter can be put together in a real experiment.

2.6.1 Frequency Response Using The Stanford Spectrum Analyser.

A White noise source can be used in conjunction with a spectrum or network analyser to test the frequency response of a loudspeaker. The experiment was performed in an anechoic chamber and a schematic of the setup can be seen below in figure 2.5.

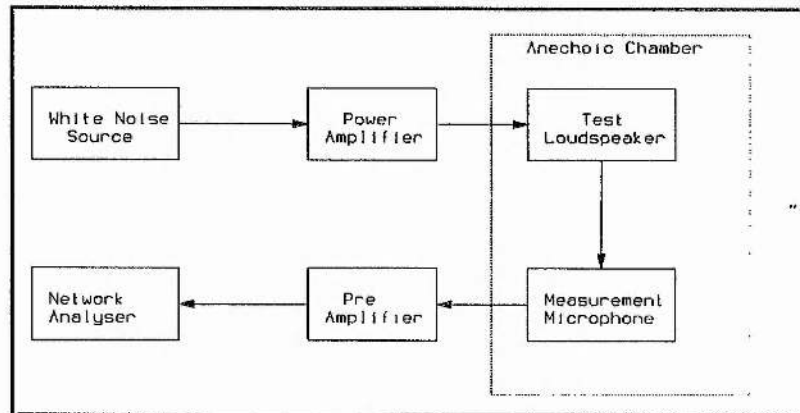


Figure 2.5: Schematic of frequency response experiment

The loudspeaker to be tested was placed in an anechoic chamber about 2m away from a measurement microphone. A pseudo random white noise source (see section 2.4) was played through the loudspeaker and the resultant waveform was picked up by the microphone. The recorded signal was sent to a preamplifier for amplification then into a Stanford spectrum/network analyser. The spectrum analyser used had a disk drive that could save data as an ASCII file on IBM formatted discs, thus allowing data to be exported to a computer. In all measurements the network analyser used 400 data bins for performing fast Fourier transforms (FFT's). The white noise source was flat to within 0.1dB between near DC and 25kHz and was assumed to be flat when being used to test loudspeakers - a good loudspeaker is typically only flat to about 5dB within this frequency range! Because of the nature of the white noise generator, it was necessary to use the averaging facility on the spectrum analyser so that a smooth response could be obtained. A typical result that

was obtained using this technique can be seen below in figure 2.6.

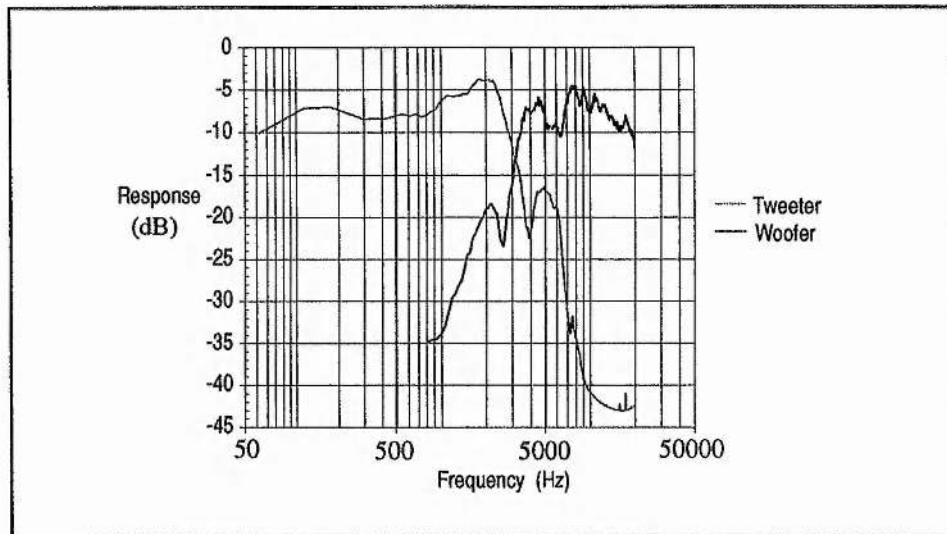


Figure 2.6: Frequency response for both drive units in a Spondor LS3 5A

Here a Spondor LS3 5A monitor loudspeaker was used as the test source. The woofer and tweeter were tested independently and both responses have been plotted on the same graph in figure 2.6. On inspection of the graph the cross over frequency can be seen quite clearly at just over 3kHz.

Although this technique worked reasonably well, it was still a long way from being ideal. The spectrum analyser always generated results made up from 400 bins, regardless of the bandwidth used. This meant that if a frequency range from near DC to 20kHz was analysed, each frequency bin would span 50Hz. The data for each bin is an average result for all of the frequencies that the bin contains. It was possible that spikes and holes at low frequencies could not be detected when analysing the whole frequency range of the loud speaker. Greater frequency resolution was obtained by reducing the frequency span on the spectrum analyser, and this quickly revealed another problem associated with the white noise generator. The pseudo random white noise generator that was being used had a repeat period of 0.131 seconds, and this caused structure to be seen at low frequencies when

smaller spans were used on the spectrum analyser. These experiments suggested that a better white noise box with a longer repeat period would need to be built. It was also realised that a stand alone spectrum analyser would not really be suitable or adaptable enough to be used for ranging experiments.

2.6.2 A Computerised Data Acquisition System.

An attempt was made to build a computerised data acquisition system to take the place of the Stanford spectrum analyser. This first attempt was unsuccessful and did not give very good results, but some very important lessons were learnt. An Acorn Archimedes computer was fitted with a Wild Vision data acquisition board. This board offered analogue to digital conversion for eight channels at up to 48kHz. This was considered a good point, because the Stanford spectrum analyser was only capable of taking one input signal at a time. At this point quite a lot of time was spent learning the C programming language and finding algorithms that could be used to perform Fast Fourier Transforms. The combination of the data acquisition board and a digital signal processing routine written in C, was intended to create a frequency analysis system which was flexible and could produce better results than the Stanford spectrum analyser.

The digital signal processing routines worked very well and were developed further. More about these routines can be found in chapter 5. The unsatisfactory results that were produced by this system, were entirely due to the data acquisition board. It was soon discovered that only one analogue to digital converter had been built on the board and it was multiplexed to each channel in turn to take readings using software. This on its own was not a big problem, however it was then discovered that the eight channels had no sample and hold circuitry. It was also determined that at high frequencies the period between successive samples on each channel was not uniform. This

was probably caused because the samples were triggered by software interrupts, and the computer processor was not always able to handle the interrupt at the exact time required. All of these factors meant that the board was unsuitable for frequency analysis experiments. It was only at high sampling frequencies that problems arose, although it was quite clear that the board did not live up to its specification. It was also frustrating because time had also been spent writing a set of library functions in assembler so that the board could be controlled from within C programs. All of the lessons that were learnt in this experiment were extremely useful when it came to designing the new system which is presented in chapter 4.

References

- ¹ Elements of Acoustics, J. Blitz, Butterworth & Co. (Publishers) Ltd, 1964.
- ² Introduction to Acoustics, R.D.Ford, Elsevier Publishing Company Ltd, 1970.
- ³ Sound, R.C. Brown, Longmans, Green and Co. Ltd, 1954.
- ⁴ Acoustical measurements, L.L.Beranek, The American Institution of Physics, 1988 Edition.
- ⁵ Fundamentals of Acoustics, L.E.Kinsler & A.R. Frey, John Wiley & Sons Inc, Second Edition, 1962.
- ⁶ Audio & Hi-fi Handbook, I.R. Sinclair, Butterworth-Heinemann Ltd, second edition, 1993.
- ⁷ Vibration & Sound, P.M. Morse, McGraw-Hill Book Company Inc, First Edition, 1936.
- ⁸ Loudspeakers for Musicians, V.Capel, Bernard Babani (publishing) Ltd, First Edition, 1991.
- ⁹ The Art of Electronics, Paul Horowitz & Winfield Hill, Cambridge University Press, second edition, 1989.
- ¹⁰ Spread Spectrum System with Commercial Applications, R. C. Dixon, Wiley Interscience, Third Edition, 1994.
- ¹¹ Digital Signal Processing Principals, Algorithms and Applications, J.G.Proakis & D.G.Manolakis, Macmillan Publishing Company, Second Edition, 1992.
- ¹² Frequency Analysis, R.B.Randall, Bruel & Kjaer Ltd, Third Edition, 1987.

CHAPTER 3.

Three Port Geometry and Modelling

3.0 Introduction

In this chapter various geometries of three port systems that can detect bearing and range are presented along with a model for synthesizing data. First a section is dedicated to providing an introduction to ranging techniques. Simple two port theory is presented to aid understanding before moving to three ports, where the theory is essentially the same although the geometry is different. In the next section, a three port geometry system from millimetre waves is presented which can be used in the near - far field. The following section looks at how a new three port geometry system was designed for use in the near field, making it ideal for use in an anechoic chamber. Finally a model is designed to enable theoretical data sets to be generated for three port systems. A series of data sets are produced, and by using the digital signal processing routines that are presented in chapter 6, the ranging capabilities of three port systems are demonstrated.

3.1 Introduction to Ranging

By using two microphones to record data from a loudspeaker under test, information such as the bearing of the source loudspeaker, or target, can be obtained. Consider the situation in figure 3.1 overleaf. Two microphones or receivers are receiving signals from a target in the far field at a certain bearing. Let us suppose that the source loudspeaker is emitting a single frequency of wavelength λ . The two receiving microphones are omnidirectional and situated a distance D apart, where $D = n\lambda$, and n is the appropriate scaling factor.

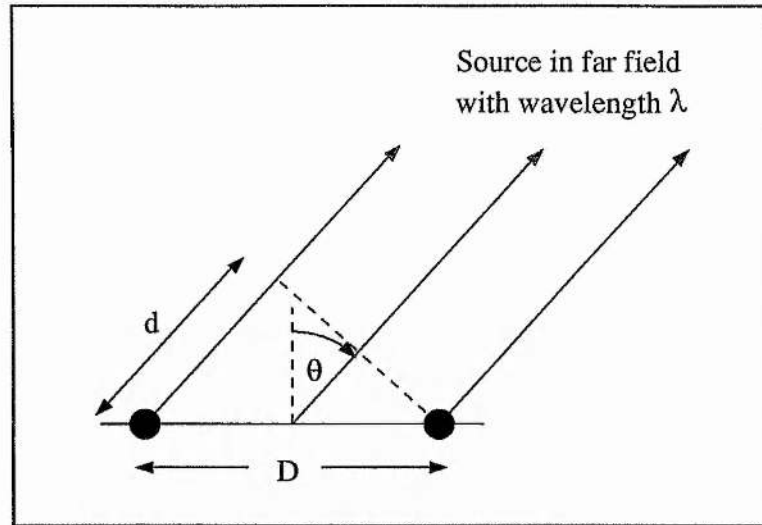


Figure 3.1: A two microphone interferometer.

The two sound waves that propagate from the source to the receivers arrive at the microphones with a phase difference that is dependent on the bearing of the source. For example, if the source is on boresight the two waves arrive in phase. When the source is at another bearing than on boresight, the two waves will arrive out of phase. The more the bearing of the source moves away from boresight, the larger the phase difference will be. If we were to combine the signals from the two receivers, the total power would fall to zero when the two waveforms are of the opposite phase. Then the path difference $d = \lambda/2$, and it is easy to find the bearing that the source has moved through when the power falls to zero. Now we can write the following:

$$d = D \sin \theta = n \lambda \sin \theta \quad \text{and} \quad d = \frac{\lambda}{2} = n \lambda \sin \theta \quad 3.1.1$$

In practice the distance d can also be found by time delaying one of the data sets received for lots of differing amounts. All of the delayed data sets are in turn superimposed on the other data set. A maximum is obtained with the data set that has been delayed the correct amount. Once the time delay has been determined,

and by knowing the speed of the wave propagation, the path difference d can be calculated. The process of applying the differing time delays to one of the data sets is in effect a means of beam steering. These techniques are typically employed in fields such as sonar and radio astronomy interferometry. There are many good books on this subject in which information with far greater detail can be found. See for example Christiansen & Högbom¹, Steinberg & Lequeux², or Thompson, Moran & Swenson³. There are two other important issues here that deserve a mention. These are field of view and resolution. The requirements for good field of view and good resolution are in conflict, so there is always a trade-off. First let us consider field of view. If a system can detect a target over a large range of bearings, it is said to have a good field of view. Suppose that the receiver spacing D was equal to half of the wavelength λ as shown in figure 3.2 below.

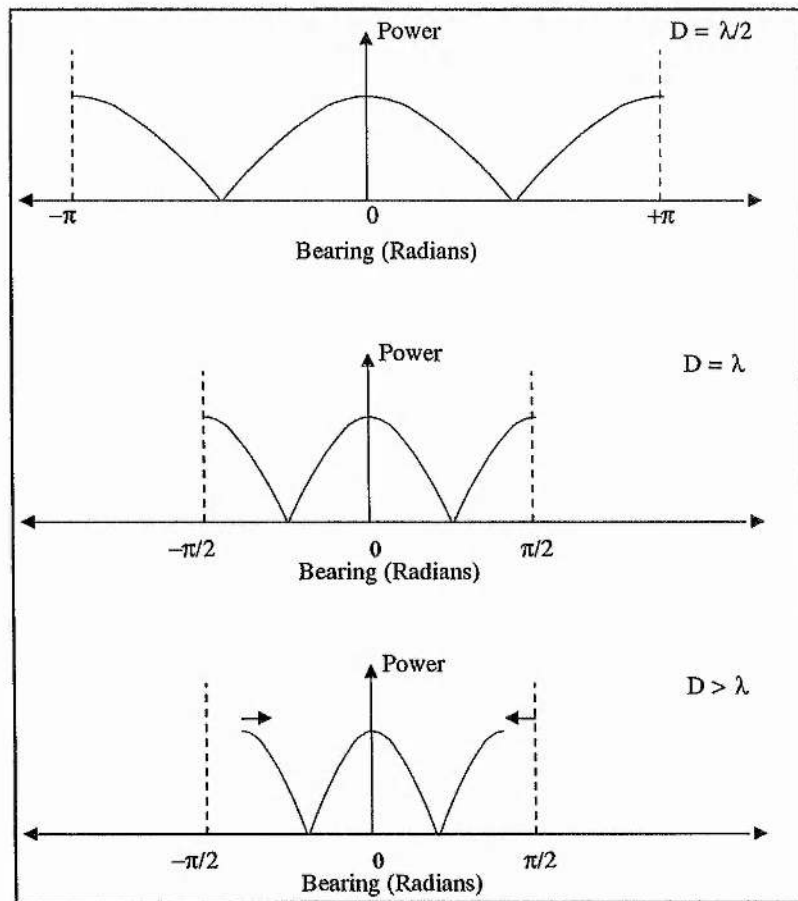


Figure 3.2: Field of View for Different Port Spacings

If, as in the top picture in figure 3.2, the receiver spacing was $\lambda/2$, the field of view would be π radians between $-\pi/2$ and $+\pi/2$. For example when the phase difference between the two receivers is $\lambda/2$, the source could be at either $+\pi/2$ or $-\pi/2$ radians. Once the source goes beyond $\pm\pi/2$ there is no way of determining which of two possible positions it is in. When D becomes bigger than $\lambda/2$, for example in the second diagram in figure 3.2 where $D = \lambda$, the field of view reduces to $\pi/2$ radians between $-\pi/4$ and $+\pi/4$. In the third diagram in figure 3.2, D becomes bigger than λ and the field of view reduces even further and so on. If more than one frequency is being used, it is the wavelength of the highest frequency that should concern us. Clearly if we were wanting to work at frequencies up to 20kHz (a wavelength of 17mm) a very small port spacing of $\lambda/2$ (8.5mm) would be impossible to obtain. Microphones have a finite size and can not really be placed accurately to such small spacings. Most of the experiments that are presented in this thesis were carried out using sampling frequencies of 24kHz, giving frequency data of up to 12kHz. Even at 12 kHz the $\lambda/2$ port spacing that would ideally be required would be 14.2mm, which is certainly not easy to achieve in practice. In reality much larger port spacings were used and the problem of multiple positions at high frequencies was overcome using other methods. This will be discussed later in this section.

Resolution is a term used to describe how close two objects can get together and still be separately resolved. An increase in resolution, or high resolution, is considered to be when two objects that are very close together can still be separately resolved, whereas a decrease in resolution, or low resolution, is when two objects need to be further apart to be able to be separately resolved. Resolution is proportional to port spacing and the wavelength, i.e.

$$Resolution \propto \frac{\lambda}{D} \tag{3.1.2}$$

Clearly we want to get the highest resolution possible and this would mean that the value for the resolution obtained in equation 3.1.2 should be as small as reasonably possible. This in effect means that we need to make the port spacing D as large as possible. This causes a direct conflict with the requirements for field of view, so normally there is a trade-off between the two.

When imaging and ranging a test loudspeaker in an anechoic chamber there are some important differences compared with ranging in sonar and radio astronomy that need to be taken into account. First, an anechoic chamber is of a small finite size and this means that any object under test will be in the near field, instead of the far field. Detectors, which in this case are microphones, have a finite size and in order that their spacings can be determined reasonably accurately, they need to be ideally as far apart as possible, thus making the spacing errors as small as possible. There are a number of sonar⁴ papers that discuss how to calibrate an array⁵⁶ with unknown perturbations, however these methods use more than two microphones for imaging in the far field and could not be easily extended for use with the three port systems that are described in the next two sections. Anything that was placed in the anechoic chamber could be put in an approximately known position, which meant that only a small field of view was required. Sources under test in the anechoic chamber were emitting white noise over a wide bandwidth which provided a very useful advantage. If the time series data from the two ports was cross-correlated in the frequency domain and then a graph of cross-correlated phase against frequency was plotted, a sort of sawtooth function would be obtained. An example of such a function can be found in figure 3.7 in section 3.4. At certain frequencies the data obtained jumps through 2π . The phase information can only be between $-\pi$ and π , so as the phase difference increases with increasing frequency there comes a point where the data jumps through $+2\pi$ or -2π depending on which side of boresight the source is on. This effect is seen to occur at higher frequencies because the field of view is very low due to the use of a large port spacing. Fortunately the white noise that the source emits covers a wide frequency

band and starts at frequency low enough where the field of view is more than adequate. As the frequency increases and the field of view decreases a continuous record of the data provides the necessary means to write a software routine in the C programming language to correct for the phase jumps and "stitch" the data back into its intended position by simply adding multiples of $\pm 2\pi$. An example of this can be seen later in this chapter in figure 3.8 in section 3.4. By using this method it is possible to use large port spacings without having to worry about the field of view.

3.2 A Three port Interferometer.

As we have seen in the last section, a two port interferometer can be used to determine the bearing of a source. By moving to three ports, enough information can be obtained to find the bearing and range of a source. The arguments about field of view and resolution that apply to a two port interferometer also apply to a three port interferometer, the main difference being only that an extra port or receiver is used. The three port geometry that is presented here is an adaptation of an idea from millimetre waves⁷ and can be used to give extra information about the source under test, such as its spatial coherence properties. In effect spatial interferometry is performed on the sound phasefront that the source radiates. The basic geometry of the three port system can be seen in figure 3.3 below. The system is based around a receiving plane that contains three ports that are linearly spaced and in which the centre port defines the origin. The source is located on the source plane which is opposite the receiving plane. The source plane could be anywhere on the circumference of a cylinder which has the receiving plane as its centre axis. This means that although a three port system can detect range and bearing, it can only do this in one plane. For example, if a target was detected a normal distance D from some point on the receiving plane, in three dimensions it could lie anywhere on a circle of radius D centred on the same point on the

receiving plane. To find the exact position of a target in three dimensions would require the use of two, three port systems, one on the x axis and one on the y axis. This would require a total of five ports with the port on the origin being shared by the two three port systems. For now we will restrict ourselves to looking at the theory with just one three port system.

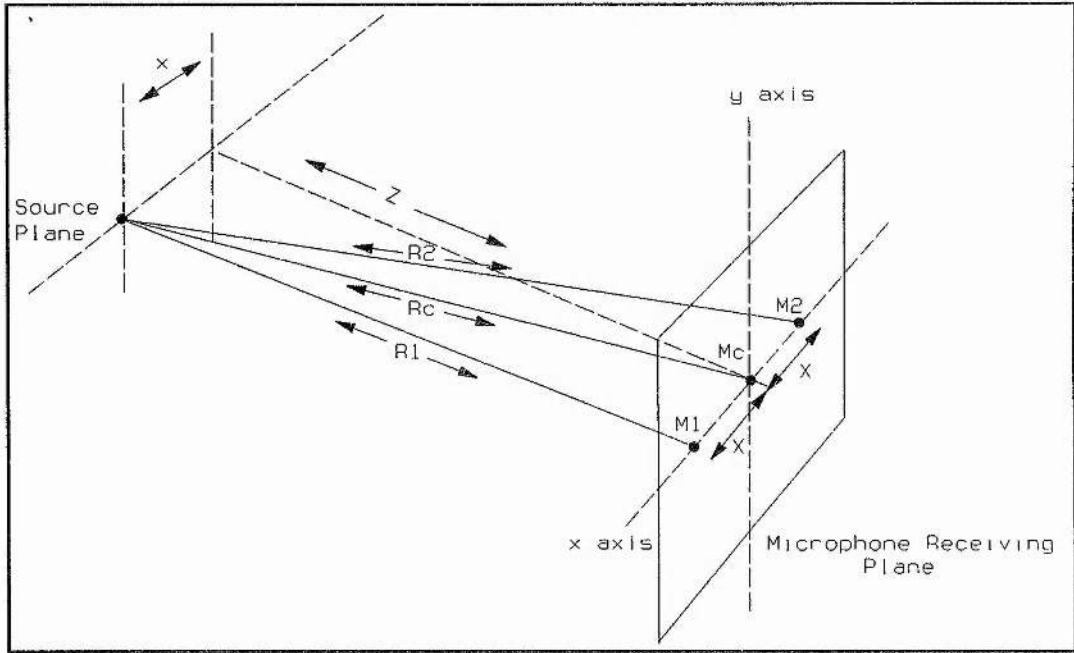


Figure 3.3: Three Port Geometry

The receiver plane can be found at a normal distance Z from the source plane and the source can be found at a distance x along the source plane. The three microphones or receivers are equispaced at a distance X . Now the system can be considered in two dimensions. The phases of the signals arriving at the three microphones will be

$$\phi_1 = \frac{2\pi\sqrt{Z^2 + (x - X)^2}}{\lambda} + \phi\{t\} \quad 3.2.1$$

$$\phi_c = \frac{2\pi\sqrt{Z^2 + x^2}}{\lambda} + \phi\{t\} \quad 3.2.2$$

$$\phi_2 = \frac{2\pi\sqrt{Z^2 + (x + X)^2}}{\lambda} + \phi\{t\} \quad 3.2.3$$

where $\phi\{t\}$ is a common phase term. Data can be taken from the three microphones or detectors simultaneously with a suitable data acquisition system. By using digital signal processing routines constructed in the C programming language we can take the FFT of the data from each channel to find the spectrum in the frequency domain. Correlation between data sets in the frequency domain can now be found by multiplying one data set by the complex conjugate of the other. This gives a discrete series of relative phase values for different frequencies. We want to find ;

$$\Theta_1 = \phi_1 - \phi_c ; \quad \Theta_2 = \phi_2 - \phi_c \quad 3.2.4/5$$

Now if $Z^2 \gg (x \pm X)^2$, i.e. if the normal distance to the source is a lot bigger than its displacement from boresight and the port spacing, it can be shown that :

$$\Theta_1 = \frac{\pi X(X - 2x)}{\lambda Z} ; \quad \Theta_2 = \frac{\pi X(X + 2x)}{\lambda Z} \quad 3.2.6/7$$

By using the series of relative phases and knowing the microphone spacing X , we can find the distances Z and x for a discrete series of frequencies or wavelengths, i.e.

$$Z = \frac{2\pi X^2}{\lambda(\Theta_1 + \Theta_2)} \quad 3.2.8$$

and,

$$x = \frac{X(\Theta_2 - \Theta_1)}{2(\Theta_1 + \Theta_2)} \quad 3.2.9$$

Therefore it should be possible to find a series of range and bearing values to locate the source of a discrete series of frequency values, thus enabling us to get a good idea of how spatially coherent our source is. It is assumed that a loudspeaker would act as a point source, although in reality at low frequencies we would expect to see dipoles. If we wanted to locate two or more sources of the same frequency we would clearly need to add more microphones to our system.

As we have seen by using this system, range as well as bearing information can be determined for given targets. This system has an added advantage in that it is passive and no knowledge of the signal the source is emitting is required. The main disadvantage with this scheme is that it is optimised to work in the middle - far field, and as we have already discussed, the environment of the anechoic chamber is relatively small and sources are really in the near field. Generally the assumption that $Z^2 \gg (x \pm X)^2$ is no longer valid. Obviously one part of the problem is that Z is small due to the size of the anechoic chamber. The other part of the problem arises because sometimes large port spacings were used to get the best resolution and field of view (as discussed in section 3.1 of this chapter). This system would have been ideal for use in an anechoic chamber if the assumption that $Z^2 \gg (x \pm X)^2$ was not needed. The next step was to look at the three port geometry again and attempt to solve it exactly. This is presented in the next section below.

3.3 An Improved Three Port Interferometer.

As we have seen in the previous section, it was necessary to solve the geometry of a three port system exactly so that it could be used in the near field and with large port spacings. A lot of time was invested in this problem and eventually a solution was found by looking at the problem in a different way. The new approach was to consider distances in terms of port to target, rather than target to port. For example, using this approach, circles can be constructed that are centred on each port with radii that are the distance to the target. With a three port system, three circles are constructed that all intersect at the position of the target - see figure 3.4 below.

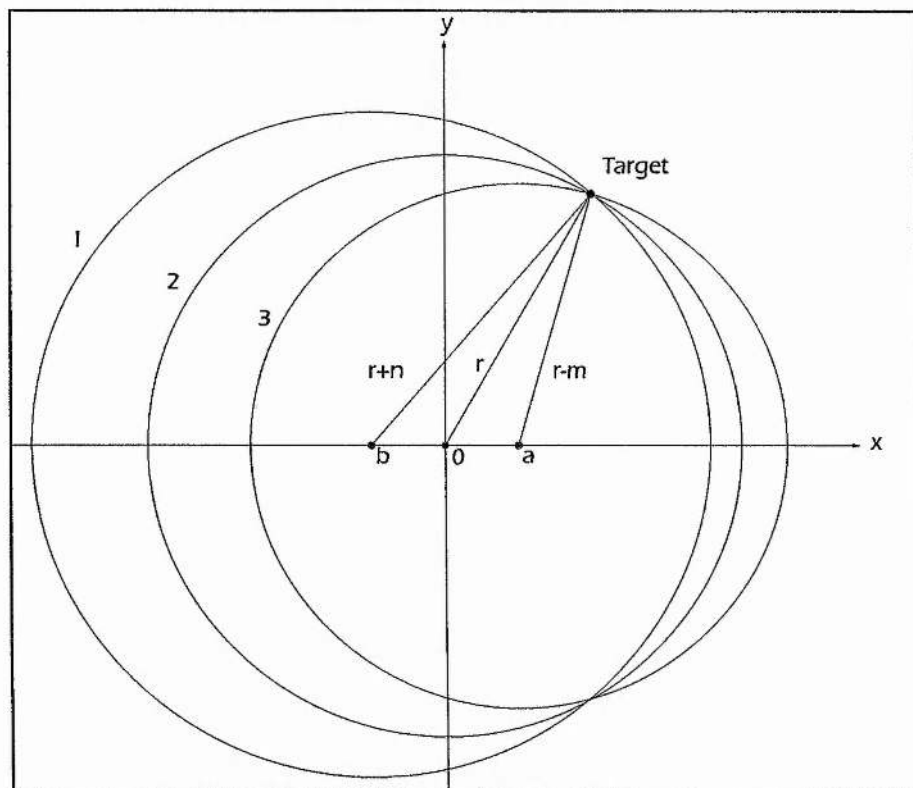


Figure 3.4: New Three Port Geometry

Here three ports are located on the x axis at b , 0 , and a . From each port a

circle can be constructed which passes through the target - circles 1, 2, and 3. Circle 2 has a radius of distance r , and it can be seen in figure 3.4 that the radii of circles 1 and 3 differ from that of circle 2 by the distances n and m respectively. The path differences m and n can be determined experimentally relatively easily, however the radius r of circle 2 can not be determined. It can also be seen in figure 3.4 that the ports that form the centre of circles 1 and 3 do not have to be equispaced from the centre port. The calculations and geometry that is presented here is for a three port system with non-linear port spacings. Although this is more complicated to calculate, the end results can easily be simplified for a three port system with linear port spacings (see section 3.3.1).

So here the objective is to calculate the co-ordinates of the target knowing only the path differences n and m and the port spacings a and b , without knowing the radius r , and without making any approximations. Let us start by considering the equations of the circles in turn.

First, circle 1,

$$(x + b)^2 + y^2 = (r + n)^2 \quad 3.3.1$$

then circle 2,

$$x^2 + y^2 = r^2 \quad 3.3.2$$

and circle 3,

$$(x - a)^2 + y^2 = (r - m)^2 \quad 3.3.3$$

We now want to use equation 3.3.2 to eliminate the radius term r from equations 3.3.1 and 3.3.3

First, substitute equation 3.3.2 into 3.3.1 to eliminate r .

$$(x + b)^2 + y^2 = x^2 + y^2 + 2n\sqrt{x^2 + y^2} + n^2$$

and then rearrange in terms of $\sqrt{x^2 + y^2}$ giving

$$\Rightarrow \sqrt{x^2 + y^2} = \frac{2bx + b^2 - n^2}{2n} \quad 3.3.4$$

Similarly equation 3.3.2 is substituted into 3.3.3 to eliminate r ,

$$(x - a)^2 + y^2 = x^2 + y^2 - 2m\sqrt{x^2 + y^2} + m^2$$

and then rearranged in terms of $\sqrt{x^2 + y^2}$ giving,

$$\Rightarrow \sqrt{x^2 + y^2} = \frac{2ax + m^2 - a^2}{2m} \quad 3.3.5$$

Now we can use equation 3.3.4 and substitute it into equation 3.3.5 to eliminate the $\sqrt{x^2 + y^2}$ term, i.e.

$$\frac{2bx + b^2 - n^2}{2n} = \frac{2ax + m^2 - a^2}{2m}$$

this can now be rearranged to give an expression for x .

$$\Rightarrow x = \frac{m(b^2 - n^2) - n(m^2 - a^2)}{2(na - mb)} \quad 3.3.6$$

Once the value of x has been determined, either of equations 3.3.4 or 3.3.5 can be rearranged to find a value for y . Equations 3.3.4 and 3.3.5 are hyperbola which vary in x and y for each possible value of the path difference (m and n) in the radii. Because the hyperbola from 3.3.4 and 3.3.5 intersect, it does not matter which of the two equations we use to find y .

So rearranging equation 3.3.4 in terms of y gives:

$$y = \pm \sqrt{\left(\frac{2bx + b^2 - n^2}{2n}\right)^2 - x^2} \quad 3.3.7$$

or by using equation 3.3.5 we can obtain:

$$y = \pm \sqrt{\left(\frac{2ax + m^2 - a^2}{2m}\right)^2 - x^2} \quad 3.3.8$$

Thus an exact and unique solution can be obtained for the co-ordinates x and y of a source or target from only knowing the port spacings a and b , and the path differences m , and n . Often in practice, linear port spacings are used, i.e. the distances from port on the origin to the other two ports is the same. When a linear port spacing is used, the equations can be simplified slightly. This situation is presented in the next sub-section that follows below.

3.3.1 Linear Port Spacings

Often in practice some redundancy is required in the array of detectors. In this particular case a linear port spacing is used. In the previous situation two separate port spacings were used represented by distances a and b . Now things can be simplified a bit by putting b equal to a .

Thus if $a=b$, equation 3.3.6 becomes:

$$x = \frac{m(a^2 - n^2) - n(m^2 - a^2)}{2a(n - m)} \quad 3.3.9$$

and equation 3.3.7 becomes:

$$y = \pm \sqrt{\left(\frac{2ax + a^2 - n^2}{2n}\right)^2 - x^2} \quad 3.3.10$$

However equation 3.3.8 remains unchanged.

Now we have a three port system with geometry that can be solved exactly and that can be used in near field experiments. This system was also ideal for use with large port spacings and no approximations were required. The geometry and calculations that have been described were used to find ranges from the results of the experiments that are presented in chapter 6. The equations were also tested by using them to find the range of data that had been synthetically generated with a modelling program. The model and the testing of the equations can be found in the following section.

3.4 Model for Data Generation

A model was designed to synthesize the data that would be received by the ports in three or multiport systems. The model was particularly useful for testing the digital signal processing algorithms described in chapter 5, as they were written and tested before the hardware for the system had been built. The model also provided a useful control function for checking that real experimental data was giving sensible results. For illustration purposes, the model for a three port system is covered here. It would be clearly possible to extend the model for more than three ports as the principle used for each port is basically the same. First, a coordinate system was defined so that all ports and sources could be written in (x,y,z) form. See figure 3.4.1 below.

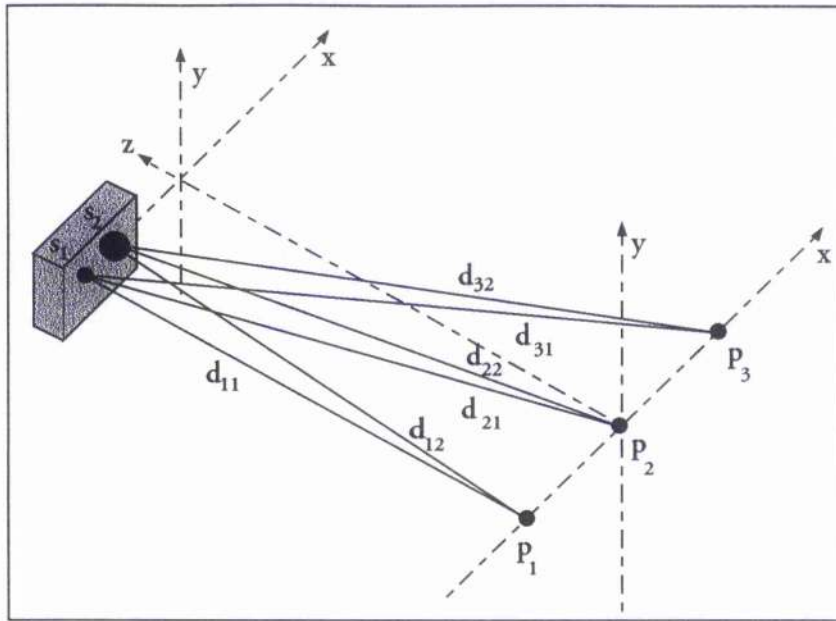


Figure 3.5: Model Coordinate System

Three ports p_1 , p_2 and p_3 were defined along with two sources, s_1 and s_2 , representing a loudspeaker. Source s_1 represented low frequencies up to a certain cross-over frequency, whilst source s_2 represented frequencies above the cross-over frequency up to half of the sampling rate. The upper frequency limit was set to be half of the sampling rate to comply with the nyquist criterion. The distances in figure 3.4.1 can be written in terms of the port and source co-ordinates :

$$d_{11} = s_1 - p_1 \quad \text{and} \quad d_{12} = s_2 - p_1$$

Similarly,

$$d_{21} = s_1 - p_2 \quad \text{and} \quad d_{22} = s_2 - p_2$$

and

$$d_{31} = s_1 - p_3 \quad \text{and} \quad d_{32} = s_2 - p_3 \quad 3.4.1$$

The C programming language was used to create the model and by using the coordinate system that has just been described, it was possible to enter co-ordinates for the ports and the sources and use the computer to calculate the values for the

the source-port separation distance. Once the source-port separation distances had been calculated, they were used to model theoretical wavefronts arriving at the ports. The objective was to model a source that was emitting white noise. A superposition of discrete frequencies was used for each sample. The two source loudspeaker model that was being used meant that the superposition was coming from two sources. Ideally, as many discrete frequencies as possible should have been used because experimentally the frequency would have been continuous, however in practice computation time meant that a compromise had to be made. For each frequency that was used a random starting phase was assigned. A series of cosine waves in the $\cos(2\pi f_i t + \phi_i)$ form were used for superposition in the model. The following equations which can be seen below were used to model data for a three port system. Each channel is presented in turn.

for channel 1,

$$\sum_{j=0}^{j=N-1} \left\{ \sum_{i=0}^{i=xf} \cos\left(2\pi f_i \left(t_j - \frac{d_{11}}{c}\right) + \phi_i\right) + \sum_{i=xf}^{i=sr/2} \cos\left(2\pi f_i \left(t_j - \frac{d_{12}}{c}\right) + \phi_i\right) \right\} \quad 3.4.2$$

similarly for channel 2,

$$\sum_{j=0}^{j=N-1} \left\{ \sum_{i=0}^{i=xf} \cos\left(2\pi f_i \left(t_j - \frac{d_{21}}{c}\right) + \phi_i\right) + \sum_{i=xf}^{i=sr/2} \cos\left(2\pi f_i \left(t_j - \frac{d_{22}}{c}\right) + \phi_i\right) \right\} \quad 3.4.3$$

and channel 3,

$$\sum_{j=0}^{j=N-1} \left\{ \sum_{i=0}^{i=xf} \cos\left(2\pi f_i \left(t_j - \frac{d_{31}}{c}\right) + \phi_i\right) + \sum_{i=xf}^{i=sr/2} \cos\left(2\pi f_i \left(t_j - \frac{d_{32}}{c}\right) + \phi_i\right) \right\} \quad 3.4.4$$

where f_i is the i^{th} frequency value that has an associated random starting phase of ϕ_i , c is the speed of sound in air, d is the distance as defined in equations 3.4.1, t_j is the j^{th} time value, xf is the cross-over frequency of the source, and sr is the

sampling rate of the system that was used to define the upper cut-off frequency and the time increments. The first term inside the curly brackets represents a low frequency drive unit with frequencies up to a defined cut-off point. The second term in the curly brackets then continues upwards from the cut-off frequency and represents the high frequency drive unit. The time increment that was used was defined as $1/sr$, i.e. the inverse of the sampling rate. Various different intervals were used for incrementing the frequency value, but generally an the smallest interval that was used was around 2Hz due to the long processing time that was involved.

As previously mentioned this model was realised using the Acorn C/C++ programming language on an Acorn RiscPc computer. A typical program written to produce three data sets of length 65536 with frequency intervals of 10 Hz took around seven hours of computer time to generate. Clearly using frequency intervals of less than 10 Hz was not a practical option when several data sets were required to be generated. The source code for the model can be found in Appendix B.

A series of data sets were generated for three ports using the model. Linear port spacings of 30cm were used for two sources. The centre port was defined as the origin. A low frequency source was defined to be (-5,0,170)cm (x,y,z format) from the origin. This source was defined for frequencies up to 3200 Hz which was considered a typical cross-over frequency for a loudspeaker. A high frequency source was defined to be (10,0,170)cm from the origin and provided frequencies from 3200Hz up to 12 kHz. The sampling rate that was used to calculate the time intervals for the data was 24kHz. After the data sets had been generated the digital signal processing routines that are presented in chapter 5 were used to process the data and range the two sources. The results from processing the data are shown here in this chapter. For a full description of the routines that are used and how they work, it is recommended that the reader consults chapter 5.

A program was written to process the synthesized data that had been generated by the model. This program was called "ProModel" and can be found in Appendix B for reference purposes. The program started by loading the data for each channel into memory. The next step was to check for any DC off-set and then to perform an FFT on each channel. A FFT algorithm of length 65536 was used the results from channel 1 of the data can be seen below in figure 3.6. Note that the power of the FFT would not normally be calculated for ranging and is has been done here for illustration purposes.

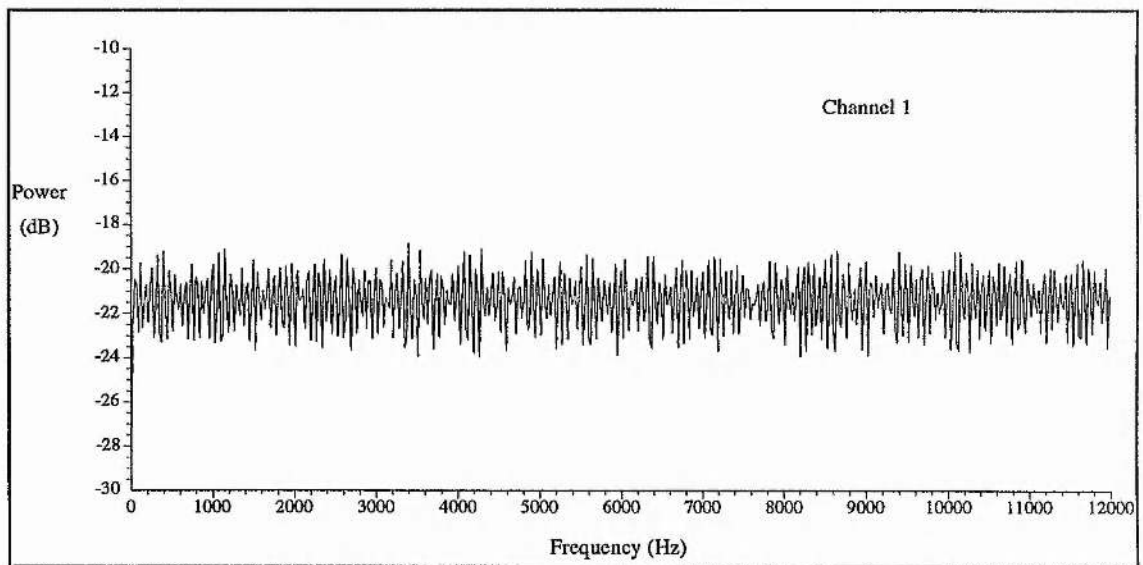


Figure 3.6 Power Spectra for Channel 1.

After each channel had been Fourier transformed, two new data sets were produced by cross-correlating channel 1 with channel 2, and by cross-correlating channel 2 with channel 3. Both of these operations were carried out in the frequency domain and gave results that were still in the frequency domain. In effect this is the same as taking the spectra of a correlation result in the time domain. The phase for the spectra of both of these cross-correlations was then calculated and the results were plotted against frequency. These results can be seen overleaf in figure 3.7. It can be seen quite clearly in the top graph of figure 3.7 that at around 3.8kHz the phase

jumps through 2π . This is a field of view problem that arises because the port spacings are large. The lower graph of the cross-correlated phase for channels 2 and 3 does not show a 2π jump because for these two ports the target is at a bearing very close to boresight and is inside the field of view for all frequencies. Both graphs in figure 3.7 show a small discontinuity at 3.2 kHz which arises due to the use of two separate sources. The discontinuity is, as expected, at the same frequency that was used as the cross-over frequency in the model.

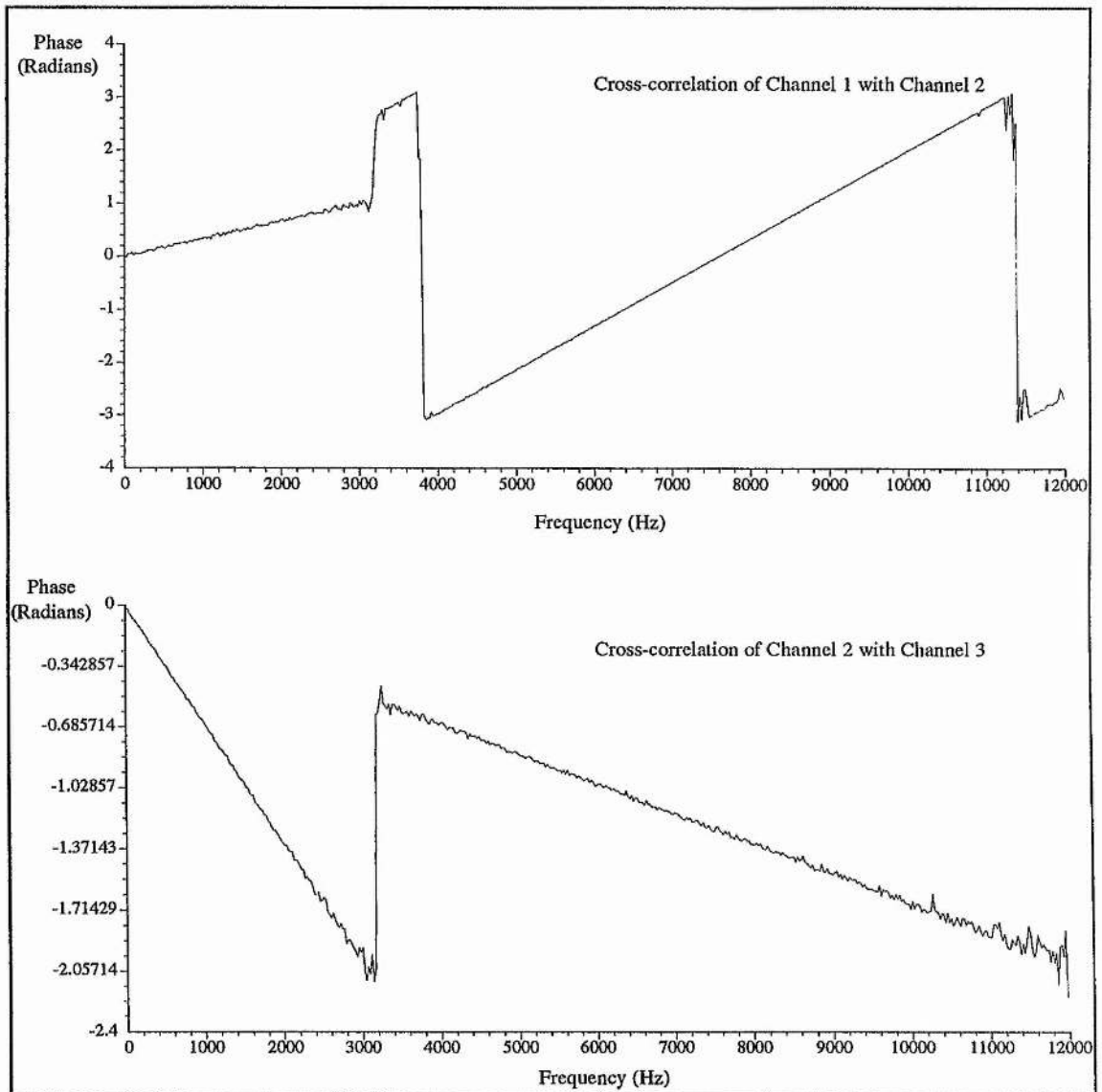


Figure 3.7: Graphs of cross-correlated phase against frequency.

The next step was for the program to check the cross-correlated phase data for 2π jumps. A routine within the program located the phase jumps and corrected the phase by adding or subtracting an appropriate multiple of 2π . Care was taken to make sure that source discontinuities were not corrected for. The results obtained after correcting the data sets shown in figure 3.7 are shown below in figure 3.8.

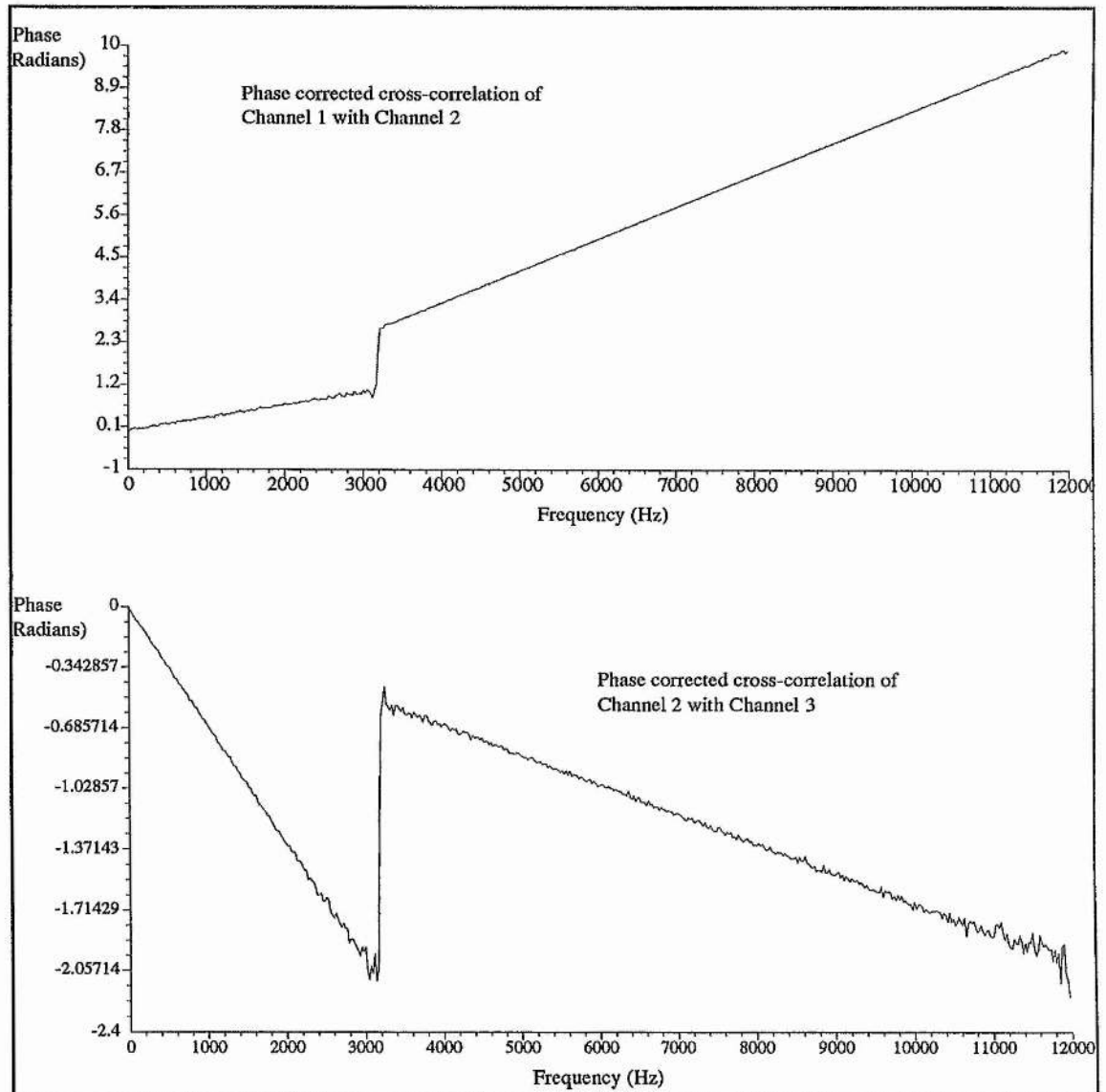


Figure 3.8: Graphs of corrected cross-correlated phase against frequency

Now it can be seen in each graph that two gradients are obtained that represent two different sources. The sign of the gradient is dependent on which channel is leading before the correlation. It can also be seen that the gradients that represent the high frequency sources would pass through the origin if they were extrapolated. This, as expected, implies that at zero frequency there would be a zero cross-correlated phase for each channel.

The two gradients on each graph were calculated using a routine that performed y on x linear regression. Once all of the gradients had been calculated they were converted to distances. We know that,

$$\phi = \frac{2\pi d}{\lambda} = \frac{2\pi df}{c} \quad 3.4.5$$

where d is the path difference between channels. Now differentiating phase with respect to frequency we have,

$$\begin{aligned} \Delta\phi &= \frac{2\pi d \Delta f}{c} \\ \Rightarrow \frac{\Delta\phi}{\Delta f} &= \frac{2\pi d}{c} \end{aligned} \quad 3.4.6$$

Here the $\frac{\Delta\phi}{\Delta f}$ term is the same as the gradients that we have obtained from the graphs in figure 3.8. Equation 3.4.6 can be rearranged and the gradient values can be used to calculate the distance d . Finally we are left with two path differences for each source. The program then took these distances and calculated co-ordinates for each source. Remember that the original co-ordinates that were used by the model were as follows,

Source	x value	z value
Low frequency	-5.00cm	170.00cm
High frequency	10.00cm	170.00cm

The results that were returned after processing the data produced by the model are shown below.

Source	x value	z value
Low frequency	-5.02cm	170.33cm
High frequency	9.99cm	169.91cm

This was considered to be reasonable proof that the model and the digital signal processing routines were working properly. More results that were created using this model can be found in chapter 6, where model data was compared with experimental data obtained.

3.5 System Characterization and Errors

As we have seen the three port system presented in this chapter works very nicely. A model has been successfully written and then used to validate the three port geometry and the digital signal processing routines. It now seemed a sensible time to perform a few more tests to determine how sensitive the system was to errors. The biggest errors in the system were likely to arise from the position of the ports not being able to be measured accurately enough. The aim of the tests presented in the following sub-sections, was to optimise the positioning of a system within the anechoic chamber so that errors due to port spacings were minimised. The tests are now presented.

3.5.1 Different Port Spacings

A new C program was written to generate a series of ranges from a three port system. The different ranges were produced by introducing a series of errors to one of the port spacings. The error in the range was recorded as a percentage error of the true range, and plotted against the percentage error in port spacing. The procedure was then repeated for several different port spacings. The results obtained can be seen below in figure 3.9.

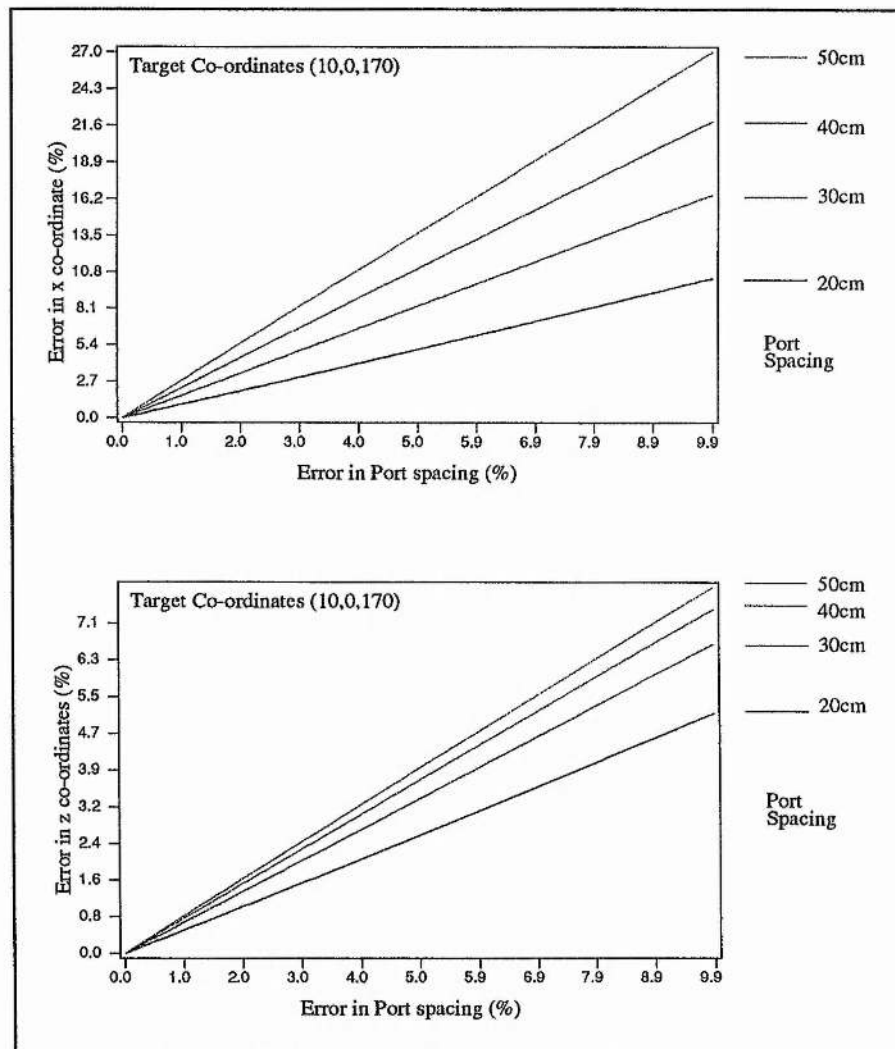


Figure 3.9: Errors in Target Co-ordinates against Errors in Port Spacings for different Port Spacings.

It can be seen on both graphs in figure 3.9 that we get a higher percentage of error in the target co-ordinates using bigger port spacings. This might be a little misleading, because in practice larger port spacings are preferable to smaller ones. It is important to remember that in reality the ports have a finite size, and they are difficult to position accurately. By using a large port spacing, the percentage error in port position is going to be a lot lower than would be obtained using a smaller port spacing. In real terms a smaller percentage error obtained using large port spacings is preferable and can give greater overall accuracy than a larger percentage error obtained using smaller port spacings.

3.5.2 Different x Co-ordinates for the Target

Another series of tests were carried out that were very similar to those described in the previous section. This time only one port spacing was used and different values for the x co-ordinate of the target were used. The z value of the target was kept constant throughout. For each position of the target a series of errors were introduced to one of the port spacings producing a series of error in the calculated target position. The results were plotted on two graphs, one showing the percentage error in the target's x co-ordinate against the percentage error in port spacing, and the other showing percentage error in the target's z co-ordinate against the percentage error in port spacing. These results can be seen overleaf in figure 3.10. Upon inspection of the data in the graphs in figure 3.10 it can be seen that the error obtained in x and z co-ordinates due to port spacing errors decreases the more the targets x value increases or moves away from boresight. It can also be noted that there is a higher percentage error in the x value of the targets co-ordinates than the z value.

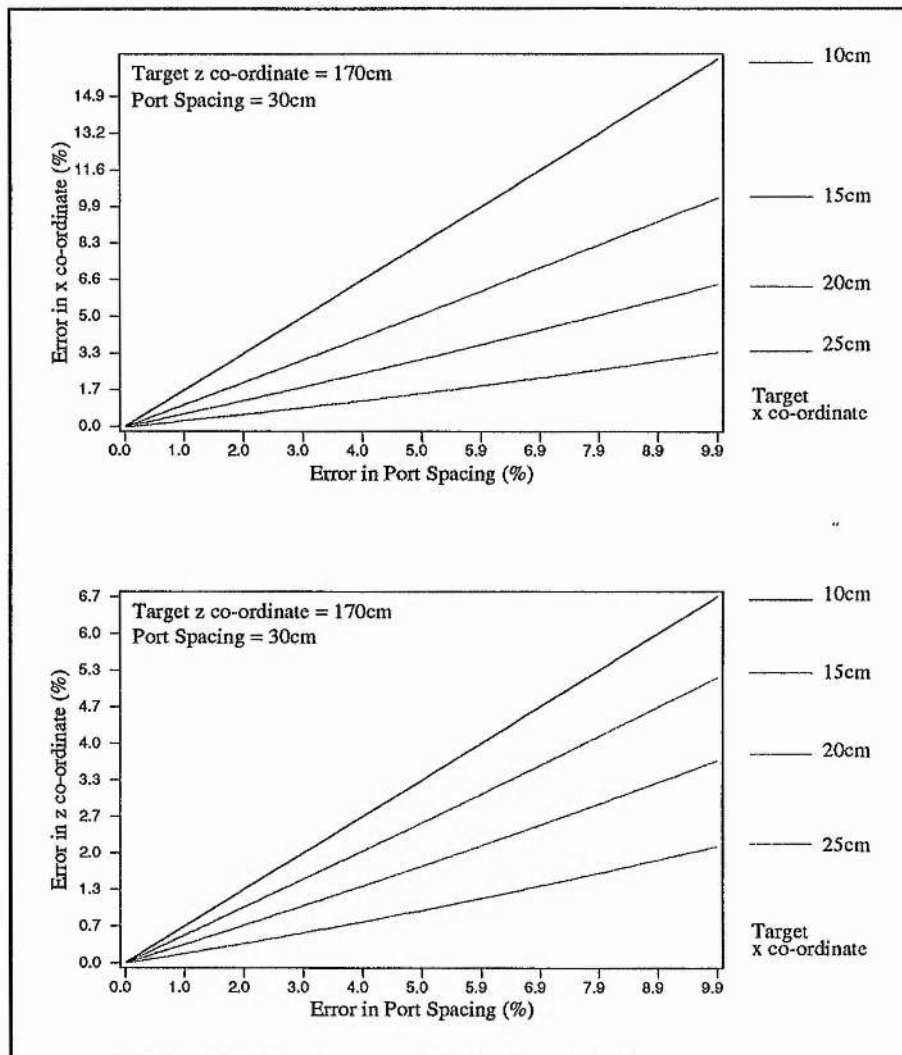


Figure 3.10: Errors in Target Co-ordinates against Errors in Port Spacings for different values of the targets x co-ordinate.

3.5.3 Different z Co-ordinates for the Target

The same tests performed in section 3.5.2 were repeated using differing z values for the targets co-ordinates and a constant x value. The results obtained can be seen overleaf in figure 3.11.

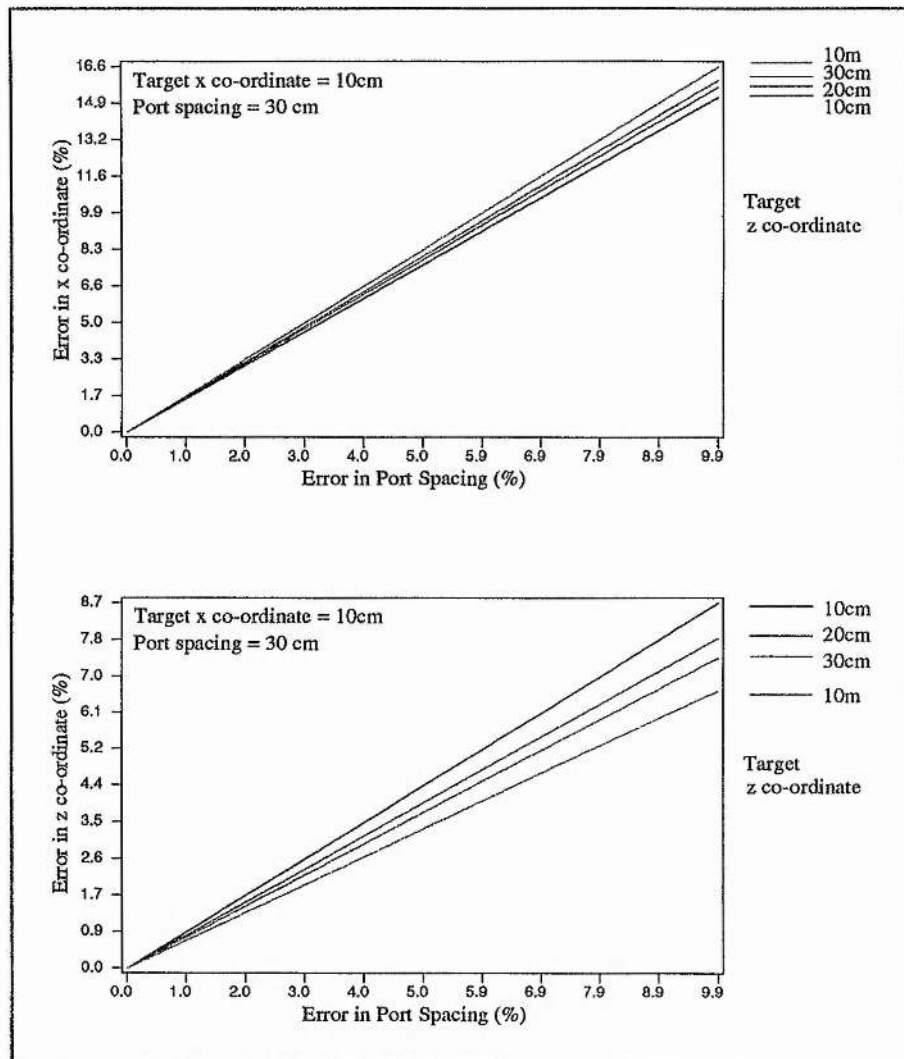


Figure 3.11: Errors in Target Co-ordinates against Errors in Port Spacings for different values of the target's z co-ordinate.

Here it can be seen that the percentage error in port spacing for a target with the largest z co-ordinate (10m), produces the biggest percentage error in x co-ordinate and the smallest percentage error in the z co-ordinate.

References

- ¹ Radiotelescopes, W.N.Christiansen & J.A.Högbom, Cambridge University Press, Second Edition 1985.
- ² Radio Astronomy, J.L.Steinberg & J.Lequeux, 1963 McGraw-Hill Book Company.
- ³ Interferometry and Synthesis in Radio Astronomy, A.R.Thompson, J.M.Moran & G.W.Swenson Jr, 1986 John Wiley and Sons Inc.
- ⁴ Applications of Optical and Radio Aperture Synthesis Techniques to Sonar, G.R.G.Erry & A.H.Greenaway, Defence and Evaluation Research Agency, 1995, Her Britannic Majesty's Stationary Office.
- ⁵ Calculation of the Shape of a Towed Underwater Acoustic Array, B.E.Howard & J.M.Syck, IEEE Journal of Oceanic Engineering, Vol. 17, No 2, April 1992.
- ⁶ Self Calibration of Towed Arrays, R.G.Marson & D.J.McLean, ISSPA 92, Signal Processing and Its Applications, Gold Coast Australia, August 16-21, 1992
- ⁷ MM-Wave Spatial Interferometry as an Alternative to radar for Coherent Point Sources, J. C. G. Lesurf & M. R. Robertson, Private Communication, 22 Apr 1993.

CHAPTER 4.

Three/Many Port System Design.

4.0 Introduction

This chapter looks at the design and building of a three/many port system. First the design requirements are discussed, based on findings and evidence from earlier experiments. The final design which was built is then presented. After an overview of the new system, its components are covered individually in separate sections. Finally information on system testing and performance is included for reference and completeness.

4.1 New System Requirements

The new system was designed with the following requirements in mind. Some of these requirements were realised from the problems encountered in previous experiments, whilst others were incorporated to allow maximum flexibility in the future :

- Noise Sources should have a repetition period longer than the data acquisition sampling period.
- Microphones and preamplifiers should be cheap, have a reasonable frequency response and ideally be as small as possible.
- A signal conditioning box should be built in the anechoic chamber control room to allow channels to be switched in or out and gains to be altered without disturbing experiments inside the chamber. This would be ideally placed just before the data acquisition system.
- The Data Acquisition System should be flexible in the amount of data it can record and have the capability to sample up to 8 channels at up to 48kHz simultaneously, preferably storing the recorded data in its own on board RAM.

Noise sources in previous experiments had caused structure to be seen at low frequencies due to a short repetition time (See Chapter 2). It now seemed a sensible idea to build a noise source with a repeat time as long as possible. This gives increased flexibility in case, for example, at some point in the future the experiment is changed to take lots of data at a low sampling rate.

In experiments up to this point, Bruel and Kjaer microphones and measuring amplifiers had been used. Whilst these are extremely convenient to use and have excellent frequency responses, they are also very expensive. The Laboratory was only equipped with two such microphones and measuring amplifiers, and due to their high cost it was considered impractical to purchase another six. For this reason new low cost microphones were sought, although it was realised that there would be a trade off between cost and frequency response. It also seemed a good idea to try and find some reasonably small microphones, thus enabling them to be positioned more accurately. Again it was realised that there would be a trade off between size and frequency response, e.g. the smaller the microphone the worse its frequency response.

A control box was required so that all channels could be switched in or out on the fly from the control room without having to enter the anechoic chamber. It was also considered a good idea to build some extra signal conditioning here, such as high pass filtering to remove any DC offset. Independent attenuators for each channel were also prescribed, thus ensuring that the full input range of any data acquisition system could be fully utilised. The other main reason for building the control box was so that it could incorporate an interface to the data acquisition system and provide a suitable place for making an earthing point for the entire system.

Data acquisition had so far been carried out using a Stanford Research Systems spectrum analyser or an Acorn Archimedes Computer with a Wild Vision analogue to digital interface board. As we have seen previously in chapter 2, the spectrum analyser could only provide a maximum of 400 frequency bins. For this reason the data acquisition was moved to the

computerised system, which in theory should have provided much more flexible data taking facilities. In practice the computerised data taking procedure was far from ideal. The main reason being that the Analogue to digital interface board had only one analogue to digital converter which was multiplexed to each channel in turn by software. Other problems arose from the cards vendor supplied software which was unreliable a high frequencies and caused sampling periods to be marred by clock jitter. All of these problems proved to be very helpful when considering how to go about finding a new data acquisition system. The obvious choice here was to stay with a computerised system but to find another A-D interface board with a better specification. The basic requirement was that the board should have eight separate A-D converters which could be sampled simultaneously, and that it should have its own on board RAM and clock so that it would be unaffected by the host computer.

4.2 New System Overview

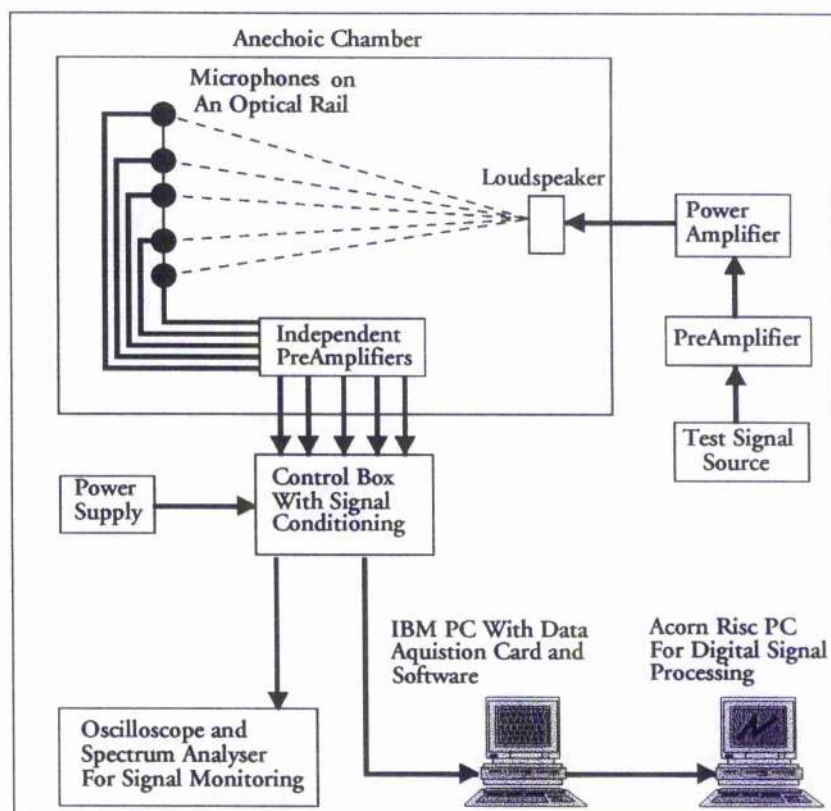


Figure 4.1 : Schematic of System.

The new system was built to accommodate as many of the requirements specified in section 4.1 as possible. A schematic of the system can be seen above in figure 4.1. Several different sources were used for producing test signals, ranging from an HP arbitrary waveform generator, a test compact disc, to custom built white noise generators. Test signals were played in to a preamplifier, which in this case was the one that was built for use with earlier experiments. A Quad 303 power amplifier was then used to drive a test loudspeaker inside the anechoic chamber. Test signals emitted from the loudspeaker were then recorded by an array of microphones located on an optical rail. Two types of microphone were used. The first type were approximately 1.5cm in diameter and had an inbuilt preamplifier powered by a small watch battery. The second type of microphone was a lot smaller, built around a microphone insert approximately 0.5cm in diameter. For this smaller type a small battery operated preamplifier was constructed very close to the insert. After the signals were received by the microphones (either type), they were amplified again by independent battery powered preamplifiers. These were constructed so that signals could be amplified to roughly 1V before making a ten meter journey to the control room of the anechoic chamber through long coaxial cables. Independent battery power supplies were used for each channel to avoid cross talk and noise problems. Outside the anechoic chamber the signals then reached a control box. This was the place where all the signal channels and inputs and outputs met. It was also the location of the systems star earth. The control circuitry provided further amplification and signal conditioning so that each channel could be optimised before being sent to the data acquisition system. An additional buffered output was provided on each channel so that signal levels and types could easily be checked with an oscilloscope or spectrum analyser. A power supply was built to power the control box. The signals were then ready to be recorded by the data acquisition system. This was an IBM compatible PC with a Fulcrum Delta Sigma DSP-Based Data Acquisition board inside one of its expansion slots. This machine was used primarily for data acquisition and the recorded data was then sent to an Acorn Risc PC for digital

signal processing. The latter machine was incorporated in to the system due to its vastly superior operating system and programming capabilities.

4.3 White Noise Sources and Other Test Signals

4.3.1. Compact Disc Test Signals

A "Sound Check" audio compact disc was used to provide some test signals. This CD provided all sorts of test tracks including spot sine waves, sweeping sine waves, white noise, pink noise and even intermodular distortion. One track provided white noise for 30 seconds at -20dB of the peak value possible on a CD. Whilst this was a good white noise source with a nice frequency response, it was not really considered practical to use as it would have meant spending a lot of time programming the CD player to repeat, and having to take data in a very short time window. Whilst the latter is possible, it would have been very inconvenient, especially when setting an experiment up and checking the attenuation levels needed to take data to utilise the full range of the data acquisition system.

4.3.2. Pseudo Random White Noise

We have already seen that a blend of analogue and digital techniques can be used to create a good Gaussian white noise source. As previously discussed, great care needs to be taken when designing the digital part of the generator in order to avoid repetitions that can cause distortion at low frequencies. In this case the digital part of the generator needed to have enough bits so that its repeat rate was longer than the longest data sample that would be taken. The original noise generator that was built for early experiments used 17 bits and was clocked at 1Mhz, giving a repeat time of 0.13 seconds. For this reason another generator was built using 31 bits with $m=31$ and $n=28$. This produced a sequence length of 2147483647 and by clocking this sequence at 1MHz a repeat period of 35.8 minutes was obtained. This new noise source was built with a trigger input so that the same noise could be used in all experiments,

if required, allowing comparisons to be drawn. More details of construction and theory of pseudo random white noise sources can be found in Chapter 2 of this thesis. Alternatively see Horowitz & Hill¹ or Dixon². The spectrum obtained with the new generator can be seen below in figure 4.2.

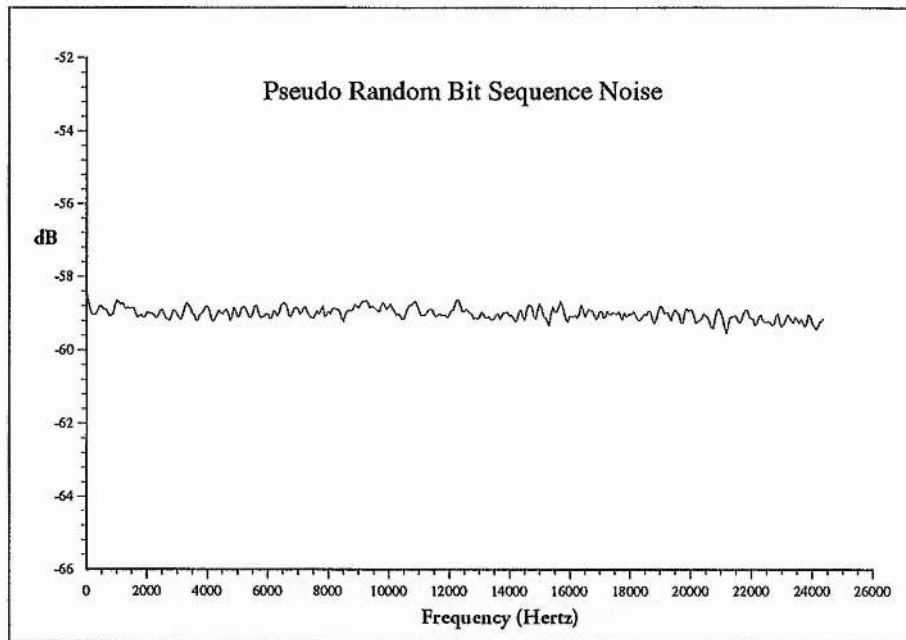


Figure 4.2 : The New Pseudo Random White Noise Generator.

(Spectrum was measured using the Stanford Research Systems Spectrum Analyser and taking 1000 averages.)

4.3.3 Transistor White Noise Source

A simple analogue white noise source was constructed using two 2N3904 transistors. A transistor with an unconnected collector is used as a zener diode in reverse bias and a second is used as a current source. The basic circuit can be seen overleaf in figure 4.3. The circuit as shown in the diagram produced a noise with a DC offset, i.e. the output voltage was between 0V and 1V. The DC offset was removed by simply adding a high pass filter between two buffers to the output of the circuit.

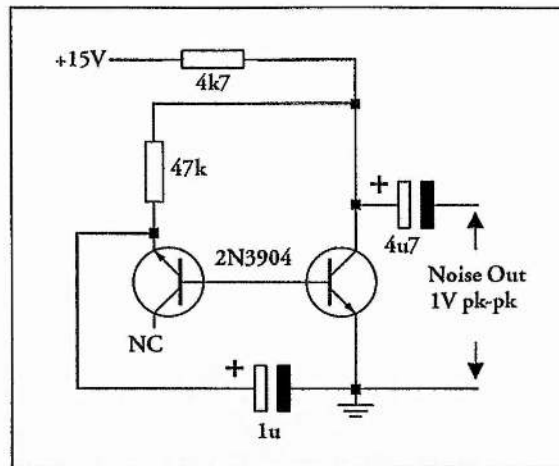


Figure 4.3 : A simple transistor white noise source.

This was a perfectly acceptable form of white noise, although in comparison to pseudo random bit sequence it had a disadvantage in that it could not be triggered. Nevertheless this source did produce a very nice flat spectrum and this can be seen in figure 4.4.

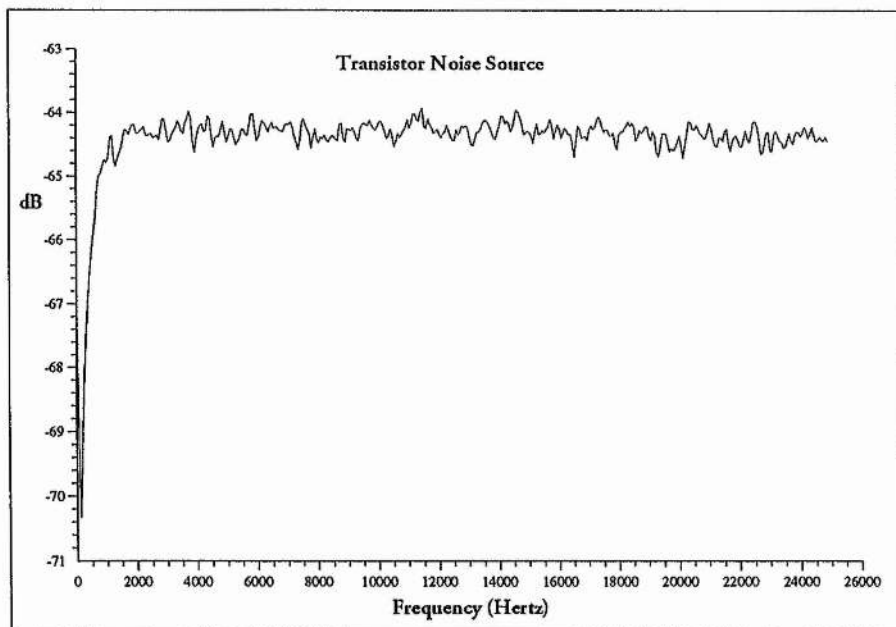


Figure 4.4 : Spectrum of a simple transistor white noise source.

4.3.4 HP Arbitrary Waveform Generator

The HP arbitrary waveform generator was extremely useful in testing and troubleshooting the system. It was particularly useful due to its digital controls, ensuring that the exact frequency and amplitude of test signals was known. Sine waves were used to test the system initially to determine correct operation and dynamic range. A second HP generator was also used to determine cross talk between channels. Although this generator was very useful for testing, the noise function was not used on an operational basis as its repeat rate was less than the longest sampling time of the system. This is the same principle as the old pseudo random white noise generator that used 17 bits.

4.4 Microphones and Microphone preamplifiers

4.4.1. Large Microphones

The Large microphones were 1.5cm in diameter and housed in a metal casing. They had a cardioid (unidirectional) response and were rated between 50Hz - 16kHz. The microphone housing was large enough so that a 1.5V watch battery cell was incorporated, providing a built in power supply. Obviously the quality of these microphones, as anticipated, was not as good as the Bruel and Kjaer microphones, but nevertheless they were considered good enough for the purposes of this experiment. A comparison of frequency responses between the different types of microphones can be seen at the end of this section in figure 4.6.

4.4.2. Small Microphones

The small microphones were based on EM-10B ultra miniature omnidirectional electret condenser microphone inserts. These inserts were 6mm in diameter and 5.2mm thick and had a frequency response of 50Hz - 13kHz. An amplifier was built into the inserts, but no power supply was provided. A power supply was built for each microphone in a small metal box and a hole was made

so that the insert could be attached with two short wires. The power supply schematic can be seen in figure 4.5a. A picture of a completed microphone can be seen in figure 4.5b and the insert can just be seen on the end of the wires coming out of the top right of the box.

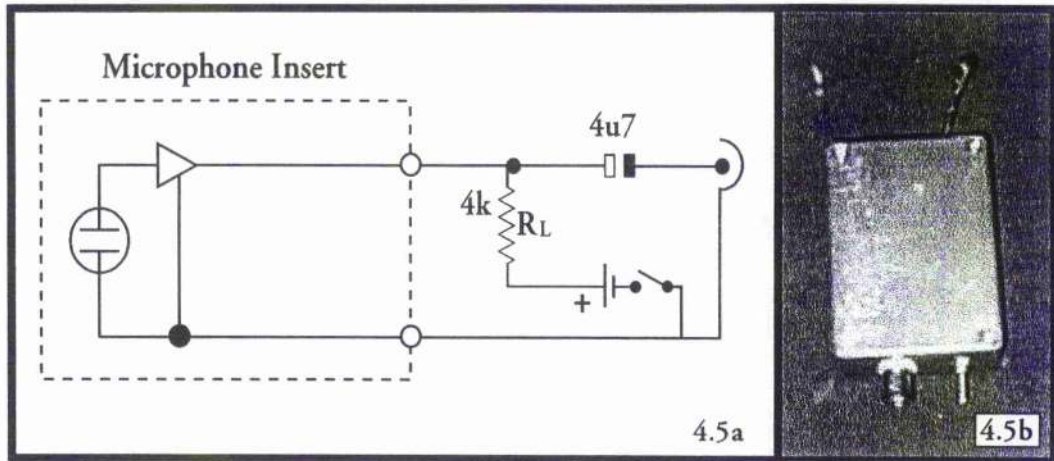


Figure 4.5 : A Small Microphone

With these smaller microphones it was expected that they would have a worse frequency response than the large microphones in the previous section. This was also reflected in the specification of the insert which was only rated up to 13kHz, probably due to its smaller size. After the microphones had been built they were tested and compared with other microphones and these frequency responses can be seen in the following section.

4.4.3. Microphone Frequency Response Comparisons

In early experiments, loudspeakers had been characterised using a high quality Bruel and Kjaer microphone and measuring amplifier. As previously discussed the Bruel and Kjaer kit has an extremely good frequency response and can be considered relatively flat. In order to discover the quality of the microphones that had been built for the new system, it was decided to use them to record the frequency response of a Spondor LS3 5A which was playing white noise. The frequency responses that were obtained were then compared with a response taken using the Bruel and Kjaer kit under identical conditions. The

responses obtained can be seen below.

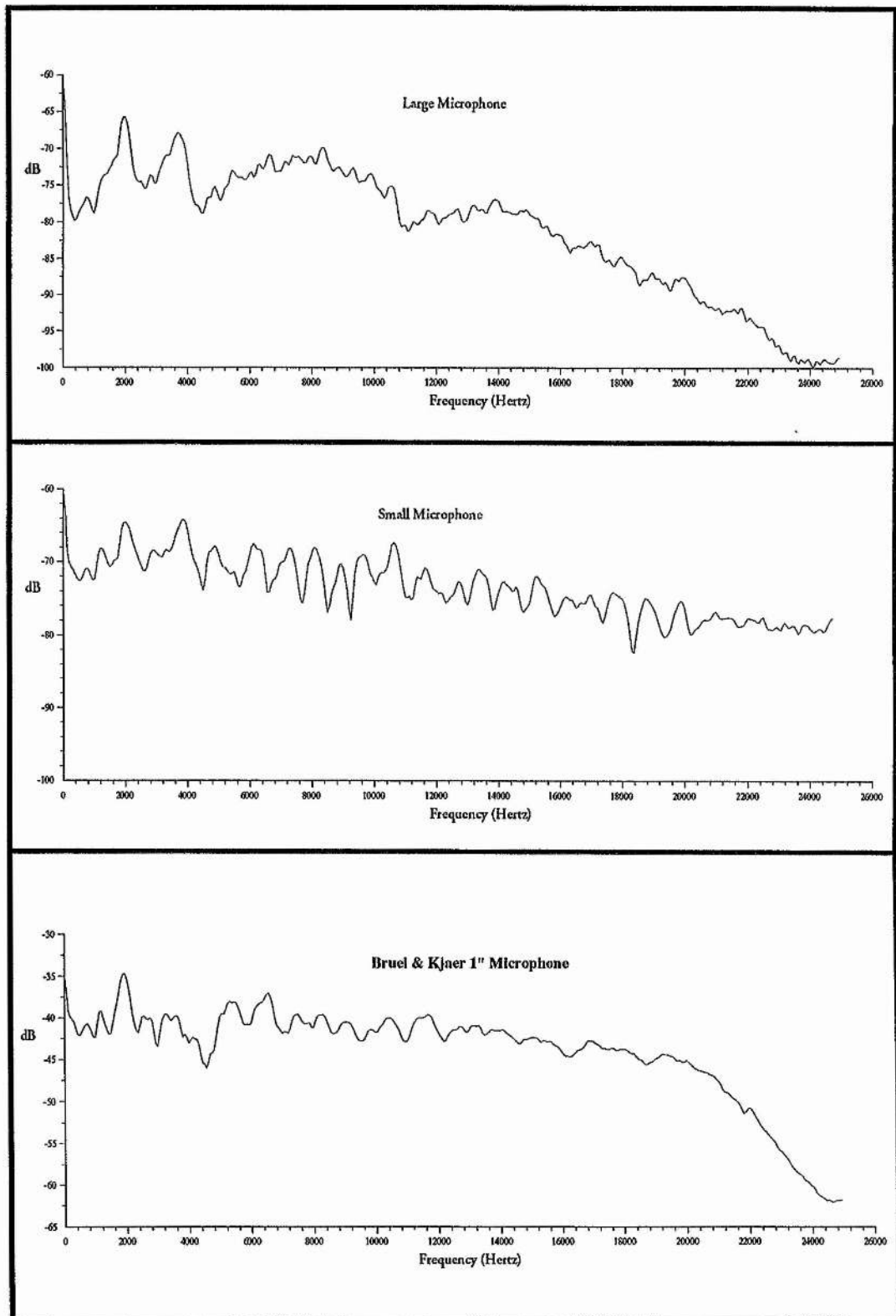


Figure 4.6: Comparison of Microphone Frequency Responses

4.4.4. Microphone preamplifiers

Given that the two different types of microphone only had output voltages of a few millivolts, it seemed sensible to build some preamplifiers to bring the signals up to about 1 volt before transmitting them down 10 meters of coaxial cable to the control room. Each preamplifier was given its own independent battery power supply to avoid cross talk and noise problems. In this case two PP3 batteries were used in series for each preamplifier. The schematic can be seen below in figure 4.7. The circuit uses two stages. A NE5534 is used as a non inverting amplifier and a LF351 is used as an inverting type. A volume control was provided by using the 10k potentiometer. Due to the nature of the circuits high gain, great care was taken at the input of the circuit to keep leads to a minimal length. Some of the later circuits were constructed on a custom designed PCB, and this can be found in Appendix A for reference purposes. A picture of five cased preamplifiers can also be seen in figure 4.7.

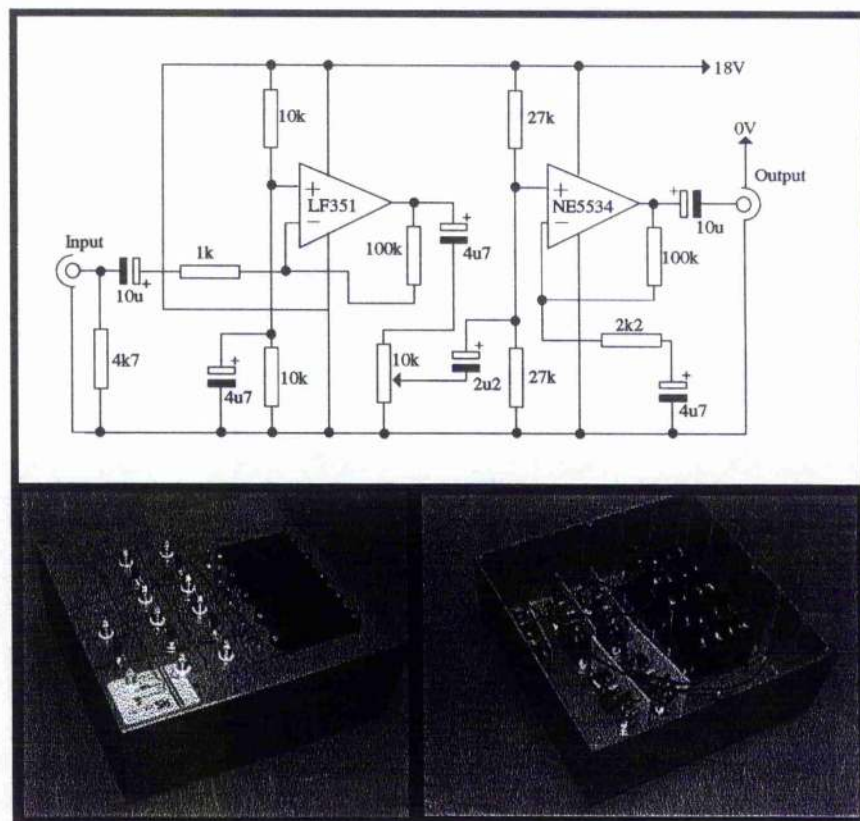


Figure 4.7: Microphone Pre-Amplifier Schematic and Pictures

4.5 Control Box Circuitry

A control box was built inside the control room of the anechoic chamber so that signal conditioning could be carried out in close proximity to the data acquisition system. A picture of the control box along with a schematic of the circuit can be seen in figure 4.8. A total of eight inputs were provided. The circuitry for each channel comprised of a high pass filter to remove any unwanted DC offset, a volume control and a gain stage. All of these stages were buffered by using TL072 OP AMPS as followers. Two buffered outputs were provided on each channel. One for the data acquisition computer and another for signal monitoring, giving a connection to an oscilloscope or a spectrum analyser. The computer output was switched so that any channels which were not being used could be switched to ground. This was a requirement of the data acquisition board. Output to the data acquisition system was provided by a 37 way D connector. The data acquisition system required differential inputs and each of these was connected to the control box. Inside the control box each channel's earth was connected together along with the earths from the differential inputs and the power supply to form one big star earth.

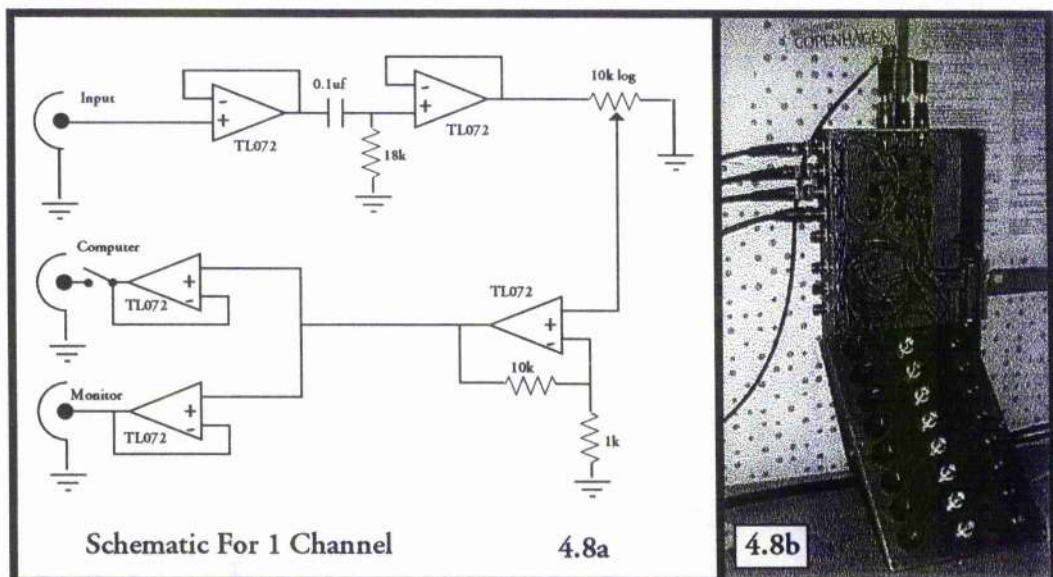


Figure 4.8 : Control Box & Schematic for 1 Channel

4.6 Data Acquisition System

Until now the term "Data Acquisition system" had meant either using the Stanford Spectrum analyser or an Acorn Archimedes computer with a Wild Vision analogue to digital board. These systems had various problems and drawbacks which have already been discussed in chapter 2 and at the beginning of this chapter. The obvious choice in this case was to find a new analogue to digital board with an appropriate specification, or to build a system and just purchase a digital input card to interface it to a computer. After doing a little research it was realised that it would be very expensive to buy an analogue to digital interface board with an appropriate specification. Indeed the cheapest available analogue to digital board cost at least £5000. The other disappointing factor here was that an appropriate board was only available on the IBM PC platform, whilst most of the authors previous work and signal processing had been designed to work on the Acorn Risc PC/ Archimedes platform. These were the only reasons that consideration was given to buying a digital input board and building the rest of the system. An adequate system, say good enough for the requirements outlined at the beginning of this chapter, could quite easily be built for less than £5000, although it would have been time consuming and probably would not have had such good performance and flexibility as a commercially available system. Ultimately a commercial analogue to digital board was purchased with financial help from the DRA (Defence Research Agency). This board was the Fulcrum Delta Sigma DSP-Based Data Acquisition System which can be seen below in figure 4.9.

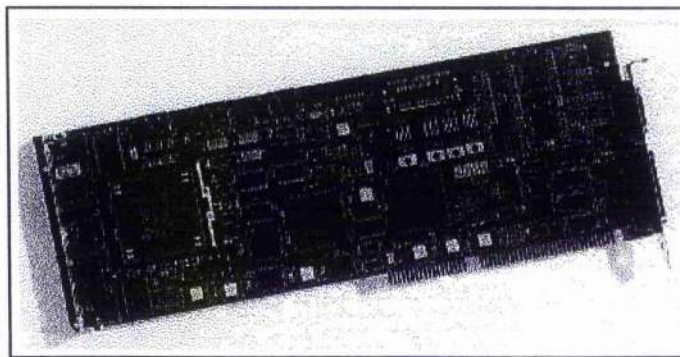


Figure 4.9 : Fulcrum Delta Sigma DSP-Based Data Acquisition System.

As the name suggests, the board makes use of Delta Sigma Analogue to Digital and Digital to Analogue data converters. In total there are eight independent A/D converters which feature simultaneous sampling (equivalent to simultaneous sample and hold). Low noise and high accuracy are achieved by using 64x oversampling on each input. Another nice feature is the built in digital filtering system to reduce errors coming from aliasing. Indeed the specification suggests that external anti aliasing filters are not needed, although this was not taken for granted. The eight A/D input channels are differential and each have a resolution of 16 bits and can be sampled up to 52kHz. Apart from the inputs, two D/A outputs are also provided featuring 16 bit resolution, and a throughput of up to 52kHz. All input and outputs are specified to $\pm 10V$.

The board also features a Texas Instruments TMS320C40 (C40) DSP engine and 4Mb of DRAM. It also has an architecture where the A/D and D/A converters are closely coupled to the C40 allowing simultaneous data collection and analysis. All A/D, D/A, and host communications are controlled by a subsystem, which can collect, process and move the data into onboard memory without interrupting the C40. The host computer is also left free to do other tasks because the data can be stored in the on board DRAM and simply downloaded as a block DMA (Dynamic Memory Access) transfer at a later time. There are also communication ports on the board, so that if more than one board is installed they can communicate and work in parallel. In theory up to 4 boards could work together in parallel. More details about this board and a specification can be found in Appendix A.

The board was installed on a PC with some difficulty. The PC was running Windows 95 as its operating system and was unable to autodetect the board. Since the board did not have drivers for Windows 95 or plug and play support, the only option was to install it by modifying all the system files by hand. This was quite difficult, time consuming and annoying. After the board had been successfully installed, the next step was to write a suitable data acquisition program.

Programming the D/A board was not a trivial task. The software

supplied was not particularly well documented and was not easy to use. A C40 compiler was supplied for use with the C programming language along with the industry standard SPOX operating system which contained math and DSP libraries. Programs for the board were written on the host PC and compiled using the C40 compiler. In operation such a program would be initiated from the host PC, downloaded onto the A/D board and then executed on the A/D board with the results being returned to the host computer at the end. Due to the difficulties in operating all the new software, only one main program was developed although a few varieties of this program emerged using different variables. The programs were written to sample all 8 channels at a predetermined frequency for as long as possible - i.e. until all of the on board DRAM had been filled up. This was done to allow ease of use, as it was easier to ignore extra unwanted data than to keep rewriting the code to produce a custom program. The sampling frequency was able to be changed quite easily by simply altering a variable in the code and recompiling it. A typical program has been included for reference in Appendix A.

After the programming had been completed, taking data was a relatively trivial task. A DOS window was opened inside Windows 95 and the data was taken by simply running the compiled program from the command line. Upon execution the program would ask for the name to be given to the output file. The data then appeared in a directory on the desktop ready for transfer to the data processing computer.

4.7 Data Processing

Data processing was carried out on a another computer instead of using the same machine that was used for the data acquisition. There were a variety of reasons for this, and more about this subject can be found in the next chapter. By using an Acorn RiscPC and the Acorn C/C++ compiler, a fast and efficient solution for the data processing was obtained. Data was transferred between the IBM PC and the RiscPC over the Internet. Initially the Linux operating system

was installed on the IBM PC so that it could be used as an FTP (File Transfer Protocol) server. The data was then imported to the RiscPC by simply opening an FTP connection to the IBM PC. This solution was not the most ideal, because it meant that after data sets had been taken on the IBM PC running the Windows 95 operating system, the machine had to be shutdown and rebooted under Linux to transfer the data. This was particularly annoying when it was necessary to process a data set before taking the next one. This inconvenience was later solved by using a shareware FTP server program that was designed to run on Windows 95. Although the two computers were in close proximity to each other, it was necessary to use the network to transfer the data due to the large amounts of data that had been generated. A typical data set was almost 4 megabytes in size and was far too big to be transferred using floppy discs. By using the FTP method a fast transfer rate of up to 300k per second was obtained - the actual rate depended on the local network traffic conditions. Once the data had been successfully transferred to the RiscPC it was processed using the digital signal processing routines that can be found in the next chapter.

4.8 Anechoic Chamber Setup

The anechoic chamber was basically set up as shown in figure 4.1. An optical rail was supported by two stools at one end of the chamber, while the test loudspeaker was placed on a stool at the other end. The microphones were mounted reasonably accurately on the optical rail using the built in scale. Generally it was most convenient to have a distance of approximately 1.7m between the optical rail and the test source due to the size of the anechoic chamber. Once an experiment had been setup, it was necessary to find the position of the source relative to the microphone that was defined to be the systems origin. This was not an easy task because of the size of the spacing and the environment in the anechoic chamber. The best results were obtained by using a reel of cotton and cutting the appropriate length off. The length could then be measured outside the anechoic chamber. Even this method was not

particularly accurate, especially when facing the fact that a loudspeaker cannot be assumed to be a point source. Another method that was considered was to use the data acquisition system to record the signal that was being transmitted by the source. This extra data could then have been correlated with the data received from the origin microphone, giving a time delay which in turn could then be calculated as a distance. It was anticipated that once the system had been built, it would be better to test it by moving the source by known positions and checking that the results showed correct differences in position. It was expected that the system would give results far more accurate than could be measured in the anechoic chamber.

4.9 System Testing

Before the system was used for the first time, the whole system was tested. A variety of different tests were made and these will now be presented. Most of the equipment such as the microphones and pre-amplifiers had already been tested and now the overall performance was largely dependent on the control box. The first test that was carried out was to switch all channels on the control box to ground. A data file was then taken by the data acquisition system. Upon inspection of the results, it could be seen that there were small DC offsets in each channel of up to 16 parts of the available ± 32768 (16 bits). Within each DC offset there was a small variation of no more than 2 to 3 parts. This test was repeated a number of times and it was quickly discovered that the offsets were random, i.e. the offset was different every time. It was decided that this problem could be best solved by writing a software routine to remove the offset after the data had been taken (see the next chapter).

The next test was made by playing a 1kHz sinewave into one of the inputs whilst keeping all the other channels set to ground. The results were processed using the digital signal processing routines and then compared with the results obtained using the Stanford spectrum analyser. Once it had been

determined that the system was working properly, another test was made to see how much cross talk there was between channels. Two HP arbitrary waveform generators were used to play sine waves in to two adjacent channels. Frequencies of 1.2 kHz and 3.9 kHz were used and the remaining unused channels were switched to ground on the control box. The results obtained can be seen below in figure 4.10.

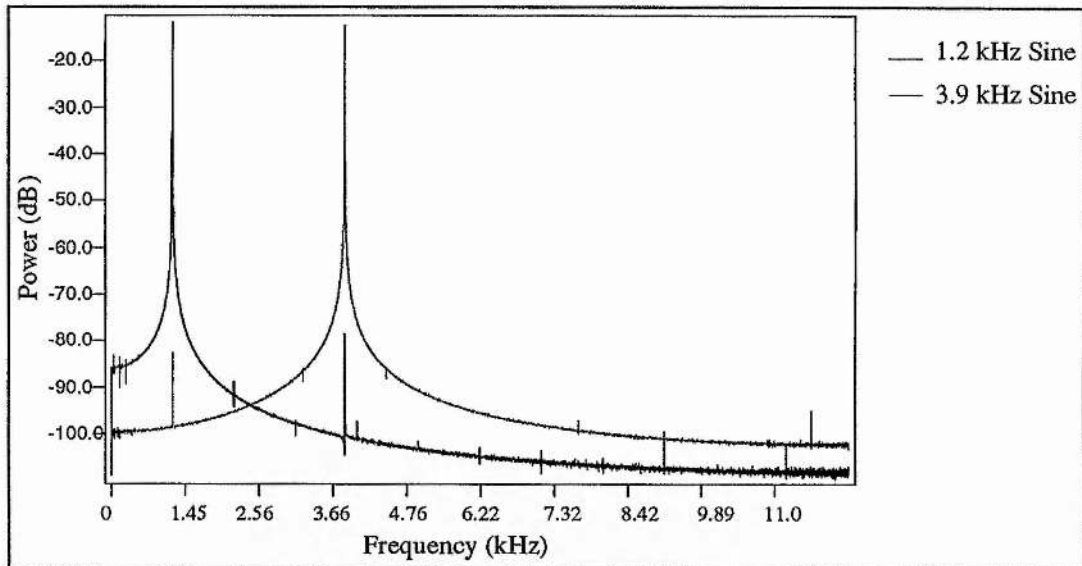


Figure 4.10: Cross-talk between two channels

It can be seen that some cross talk was obtained between the two channels. In both channels the cross talk is at -80 dB or below for signal levels of -10 dB. Although this cross talk was not desirable, it was considered not to cause significant problems in the system. Once the tests had been completed the system was considered to be ready for operational use. Results obtained using the system can be found in chapter 6.

References

- ¹ The Art of Electronics, Paul Horowitz & Winfield Hill, Cambridge University Press, second edition, 1989.
- ² Spread Spectrum System with Commercial Applications, R. C. Dixon, Wiley Interscience, Third Edition, 1994.

CHAPTER 5.

Digital Signal Processing.

5.0 Introduction

A considerable amount of time and effort was invested in designing the digital signal processing algorithms described in this chapter. Throughout the last two years of this thesis study, these algorithms were used on an almost daily basis. It is worth pointing out that the use of most of these routines was made possible by recent advances in processor technology and memory price, making the processing quick and affordable. In this chapter, issues such as choice of programming language and hardware platform are discussed. The routines described here were written in the C programming language for the Acorn RiscPC platform. Most of the routines were written, compiled, and then incorporated in to library functions. The routines in each of the library functions are presented in turn. The structure of the routines and how the algorithms work is discussed, however the full source code has been omitted and can be found in appendix B at the end of this thesis. Most of the algorithms were calibrated and tested. The resulting performance is discussed in the conclusion of this chapter.

5.1 Hardware and Software Issues

5.1.1 Platform Choice

Although an IBM PC compatible computer was used for collecting data, it was not used for processing it. Initially, it was decided to try and use the IBM PC for convenience reasons, although this eventually proved to be an impossible task. The main problem was related to memory allocation when writing data processing algorithms in the C programming language. A Borland C/C++ compiler was used under the Windows 95 operating system,

and this was considered to be a good and industry standard compiler. However, whilst the compiler offered many useful features for making windows applications, it did not have the capabilities to allocate large arrays of memory. In one sample, the data acquisition system generated almost 4 megabytes of data. Clearly this would require large amounts of memory to be allocated if the data was to be processed quickly and efficiently.

Unfortunately it became impossible to allocate double precision arrays over 65536 cells long. This was due to the fact that integers or unsigned integers were the only types that could point to an array, and they were both defined using 2 bytes (16 bits). Quite a large amount of time was invested in trying to find a way around this problem, but in the end the decision was made to go back to using the Acorn platform that had been used in previous experiments. Although the Acorn C compiler did not come with as many features or as much documentation as the Borland compiler, it did have the ability to create larger arrays. In the Acorn compiler, both integers and unsigned integers were defined using 4 bytes and this meant that massive arrays could be allocated with relative ease and were only limited in size by the amount of memory in the computer. A quick comparison of how different the different types varied between the two compilers can be seen in the table below.

Type	Borland C	Acorn C
int	2 bytes	4 bytes
unsigned int	2 bytes	4 bytes
long	4 bytes	4 bytes
float	4 bytes	4 bytes
double	8 bytes	8 bytes

By using a different computer for doing the data processing the additional inconvenience of transferring the data arose. As we have already seen in the previous chapter, this was accommodated by using File Transfer Protocol (FTP) between the two computers over the Internet. It was unfortunate that a

suitable data acquisition card was not available for the Acorn platform, as this would have been the ideal solution. An Acorn RiscPc PC with a Strong Arm processor machine and 50 megabytes of memory was used for all of the digital signal processing, and to present the results. The main advantage of using a computer of this type was realised in terms of speed and ease of use. All in all the RiscOs operating system was far superior in comparison to the windows operating system.

5.1.2 Programming Language Choice

The C programming language was chosen to create all of the data processing routines. This language was chosen primarily because of its speed in performing double precision floating point operations. Given the amounts of data that needed to be processed, and that it would need to be fast Fourier transformed, meant that there was no real alternative to using C. C is a low level language which is very powerful, yet at the same time can be very difficult to program as there are no real error warnings or support that one might find in a language such as Basic. At the beginning of this Ph.D. study the writer invested a lot of time in learning how to program in C, and it did indeed prove to be very useful. Initially just the basic C compiler was used. Programs were written as one piece of code and then compiled. Later, after gaining more experience with C, programs were written as lots of small pieces of code, making use of library functions and compiler tools such as Make. Routines that were used often and in many different programs were written as library functions and compiled so that they would not need to be rewritten each time that they were needed. Such library routines could then be included and called from any other source code. All programs were then constructed using the Make project management tool. In effect this was an automated way to individually compile all the different source code needed in a project and then create a final program using the Link tool. By using these techniques, the main part of programs were very compact and relatively easy to debug. This also meant that it was very easy to write new programs in a

matter of minutes, especially when the bulk of the work could be accessed from library functions. For more details on the C programming language see Kelley & Pohl¹ or any other of the many available books on C.

5.1.3 Memory Allocation

As we have seen in section 5.1.1, memory allocation was quite a big issue, however, it was not only an issue limited to the choice of platform. Whilst using the Acorn C compiler some additional problems had to be overcome. One problem was preserving the data in large arrays. Sometimes when large arrays of data were allocated, the data became overwritten before the program had finished running. This meant that programs would sometimes give meaningless results and often the program would crash. It was noticed that this seemed to be a random problem. Sometimes the program was unable to allocate the required memory depending on the other tasks the computer was doing. Memory had simply been allocated by using a simple command such as `double array_name[131072]` and it was eventually realised that this was an unreliable means for creating very large arrays, although it did seem to be perfectly satisfactory for small arrays. This problem was solved by using the `malloc` command to allocate memory, and although this meant writing a few more lines of code it worked very satisfactorily. It was also realised that great care was needed to be taken to deallocate the arrays when any program finished, otherwise the computer would have soon run out of memory. More about this command and the library function that was written using it can be found in section 5.5. The other issue was related to how much memory the program took while it was running. Initially, before extra RAM was purchased, the RiscPc PC only had 18 megabytes of RAM. Some programs quickly filled this space and errors were reported because the `malloc` command was unable to allocate the required memory. Although 18 megabytes seemed a lot of memory, by the time a 4 megabyte data set had been loaded and processed it could be very quickly filled. It is also worth noting that a double precision floating point variable requires 8 bytes of

memory to be stored. Once this had been realised, programs were written so that memory arrays were allocated and deallocated as and when they were needed during the course of the program, instead of being allocated at the beginning and deallocated at the end. By using this technique most programs were able to run inside the available memory. Later an additional 32 megabytes of memory were purchased so that programs could be written to process data for more than three ports without having to worry about running out of memory.

5.1.4 General Software Issues

Care was taken to scale input data and checks were made throughout programs to make sure that precision was not lost during the signal processing operations. Another problem which arose during calculations was division by Zero. If this event was not checked for and occurred during digital signal processing, the program would abort and return with an error message. With real data this situation did not occur frequently, and it was not considered necessary to trap these possible errors. One other subject which deserves mention here is that of pointing to arrays. Throughout the rest of this chapter the library functions that are described take input data into arrays defined between 1 and some number N . Any person with a good knowledge of C will know that arrays created using the malloc command are defined between 0 and $N-1$. Data was passed between these two different array types by using an appropriate pointer.

5.2 File Library (Filz)

A library called Filz was written to provide functions to input data from the data acquisition system and to output results to disc. The data acquisition system gave an output of two byte integers in two's complement binary data format. A suitable routine was required to load this data into an

array of memory on the RiscPc. It was decided that the RiscPc had enough memory to justify loading the whole data set in to memory, regardless of how many of the eight available channels were being used. This was a very useful decision and it meant that only one data loading routine would ever be required. Any data from unused channels was then disregarded in the program. In C there is a standard library that provides functions for reading from and writing to disc. Whilst these functions are very useful, they are also relatively slow. Given that over 3 megabytes of data was being loaded, a quicker alternative was sought. Fortunately the RiscOs operating system had a command for loading blocks of data in assembly language. A routine was written to access this command from within C and load the data. The structure of the function is shown below.

```
void Load_Data(int data[])
```

This routine loaded data from the hard disc and put it inside an array called data. The routine always looked for a input file called "test1" in a predetermined directory and was capable of loading as little or as much as the data was required. Data was produced by the data acquisition system in the Ch1 - Ch8, Ch1 - Ch8, etc. format, so in practice usually all of the input data was read by the routine. The code for the routine can be found in appendix B for reference. Normally after this routine had been used, the integer data was scaled to provide a double precision floating point data set in the range of plus or minus 1.

Two other routines were included in this library to make the output of data to a file a trivial task. Although the RiscOs operating system had an assembly command for saving blocks of data, it was decided to use the C language's data output function because it gave more flexibility. It was also reasonably slow, but in general only small amounts of processed data were to be saved to disc. The routines which can be seen below were designed to

produce comma separated variable (CSV) output files so that the data could then be easily loaded into a graph drawing program to display the results.

```
void Ram_write(double data[],const char fname[],int start_point, int end_point, double xinc)
```

This routine produces a CSV output file in X,Y format. The Y values come from the array `data[]`, and the X values are calculated using the `start_point`, `end_point` and `xinc` input variables. The variables `start_point` and `end_point` were used to point to a section of the array `data[]` that was being exported, whilst the variable `xinc` was needed to scale the X values. It follows that any part of the data set could be saved by choosing appropriate values for `start_point` and `end_point`. The other routine is now shown below.

```
void Ram_write2(double data[],double data2[],const char fname[],int start_point, int  
end_point, double xinc)
```

This second routine is almost identical to the first, except that it takes two input arrays, `data` and `data2`, and produces a CSV output file in the `X1,Y1,X2,Y2` format. This routine was occasionally modified so that more than two channels could be output. This has not been included here as the modifying the routine should be a trivial task. Both of these routines can of course be found in Appendix B.

5.3 Digital Signal Processing Library

This library was used to house the routines that performed the digital signal processing. Two routines can be found for performing a FFT along with one for cross correlations. All of the routines perform calculations on

double precision floating point data and were used for processing the data from most of the experiments. Other routines were written and tested, but were generally inferior to the ones in this section. It would not have been obvious if the routines were not working properly so they were tested to ensure correct operation (See section 5.7). The routines are now presented in the following sub-sections.

5.3.1 Fast Fourier Transform (FFT) Algorithm

There are many advantages in using the FFT algorithm over the discrete Fourier transform (DFT). A DFT of N points can be written as follows:

$$H_n = \sum_{k=0}^{N-1} W^{nk} h_k \quad 5.3.1$$

where

$$W \equiv e^{2\pi i/N} \quad 5.3.2$$

This algorithm is a process of the order N^2 . By utilising a FFT algorithm the DFT can be changed in to a process of the order $N \log_2 N$. Clearly this can make an immense difference when N is large! Danielson and Lanczos showed that a DFT of length N can be rewritten as the sum of two DFT's of length $N/2$. The proof is as follows:

$$\begin{aligned} F_k &= \sum_{j=0}^{N-1} e^{2\pi i j k / N} f_j \\ &= \sum_{j=0}^{N/2-1} e^{2\pi i k (2j) / N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i k (2j+1) / N} f_{2j+1} \\ &= \sum_{j=0}^{N/2-1} e^{2\pi i k j / (N/2)} f_{2j} + W_k \sum_{j=0}^{N/2-1} e^{2\pi i k j / (N/2)} f_{2j+1} \\ &= F_k^e + W_k F_k^o \end{aligned} \quad 5.3.3$$

The two new DFT's are formed, one from the even numbered points of the original DFT and the other from the odd numbered points. The good thing

about implementing Danielson-Lanczos Lemma is that it can be used recursively. For example, one of the new DFT's formed from the even numbered points of the original data, can be broken into another two DFT's formed from the even-even and even-odd points of the original data. This technique means it is most convenient computationally to calculate FFT's where N is an integer power of two. The result of using the FFT algorithm is that the data becomes subdivided into lots of transforms of length 1. These transforms can be solved easily with the help of a bit reversal algorithm, thus making the FFT practical. So the structure of the FFT algorithm is in two parts. The first part sorts the data into bit reversed order, while the second implements the Danielson-Lanczos Lemma calculating transforms of length 2, 4, 8, ..., N in turn.

The C code routine for implementing the FFT has been called `dfour1` and its structure can be seen below. It is based around the "Numerical Recipes in C"² routine `four1`, which is a single precision floating point routine.

```
void dfour1(double data[], unsigned long nn, int isign)
```

If `isign` is 1, `data[1...2*nn]` is replaced by its discrete Fourier transform; or if `isign` is -1, `data[1...2*nn]` is replaced by nn times its inverse discrete Fourier transform. In the latter case the data will need to be multiplied by a normalizing factor of $1/N$. Here the data is a complex array of length nn . In this routine nn must be an integer power of two and it is important to note that this is not checked for in the code. The code for this algorithm is not shown here, but is included for reference in Appendix B. The data input and output array formats are shown in figure 5.1 overleaf.

Time Domain		Frequency Domain	
Array Cell No	Time	Array Cell No	Frequency
1 real 2 imag	$t = 0$	1 real 2 imag	$f = 0$
3 real 4 imag	$t = \Delta$	3 real 4 imag	$f = \frac{1}{N\Delta}$
etc etc	etc etc	etc etc	etc etc
etc etc	etc etc	N-1 real N imag	$f = \frac{N/2-1}{N\Delta}$
etc etc	etc etc	N+1 real N+2 imag	$f = \pm \frac{1}{2\Delta}$ (Combination)
etc etc	etc etc	N+3 real N+4 imag	$f = -\frac{N/2-1}{N\Delta}$
2N-3 real 2N-2 imag	$t = (n-2)\Delta$	etc etc	etc etc
2N-1 real 2N imag	$t = (N-1)\Delta$	2N-1 real 2N imag	$f = -\frac{1}{N\Delta}$

Figure 5.1 Data input & output array format.

All of the data which has been experimentally generated is real. This would mean that all the imaginary parts of the array used in the routine `dfour1` should be set to zero. Clearly this is inefficient in terms of processing time and the amount of memory storage space required. There are two methods for overcoming these two problems.

The first is to pack two independent real functions into one input array. We know that because the input data of the transform is real, the components of the DFT satisfy

$$F_{N-n} = (F_n)^* \quad 5.3.4$$

where the asterisk indicates the complex conjugate. Similarly the DFT of purely imaginary set of data has the opposite symmetry e.g.:

$$G_{N-n} = -(G_n)^* \quad 5.3.5$$

By packing one real data set into the real part of the input array and another real data set into the imaginary part of the input array the data can be transformed by the routine `dfour1`. The transform array which is produced can then be unpacked into two complex arrays by using the two symmetries. There is however one disadvantage of using this method, and that is that it is not particularly convenient to use if one we only have one data set which we wish to transform and this issue is addressed in the second method.

The second method is to pack one real data set into a complex array of half the size. This can be achieved by splitting the original real data into two sets, one formed from the even numbered data and the other formed from the odd numbered data. The first set becomes the real part of the transform array whilst the second set becomes the imaginary part. The data can now be transformed as described in the first method, however there is a difference in this case. The result will not be the transform of the original data, but a combination of two transforms each containing half of the required information. This is treatable and is used in a routine called `drealft`. The form of the routine structure is shown below. The method for rearranging the two jumbled data sets is not covered here as it is considered to be outside the scope of this thesis, however a more detailed description can be found in "Numerical Recipes in C"².

```
void drealft(double data[], unsigned long n, int isign)
```

The above routine calculates the Fourier transform of n real data points. The source code for this function can be found in Appendix B for reference. It should also be noted that this routine is dependent on `dfour1`, i.e. it calls the function `dfour1` when it is used. The input data inside `data[1..n]` is replaced by the positive frequency half of its complex Fourier transform. This is quite convenient as the transform results fit exactly into the input array. Because of equation 5.3.4 the negative frequency half of spectrum can be discarded, thus

avoiding having to save all of the data. One problem that arises here is that we need spectrum values from $n=0.....N/2$ and in fact we only have $N/2$ available spaces in the working array. In this case, the fact that first and last values of the complex spectrum are real and independent allows the problem to be solved by placing them in the output array as `data[1]` and `data[2]` respectively. In this routine n must be an integer power of two, as this is a requirement in `dfour1` which is used to perform the transform. The process is also reversible by replacing 1 with -1 for `isign`. In the case of the reverse operation, the data must be multiplied by a normalizing factor of $2/n$.

The routine `drealft` was chosen as the most suitable routine to perform Fourier transforms. Although it involved using a little more code than just using the routine `dfour1`, it was far easier to implement, saving memory space and maximising computational efficiency. This method was also chosen over that which performs two real FFT's simultaneously, because in practice one may have an odd number of data sets. Thus an effective quick and efficient routine for Fourier transforms had been developed. By writing the routines (This means routines `dfour1` and `drealft` since the later is dependent on the former) as library functions, it also meant that transforms could be made by just using one line of code in the main program. As discussed in the introduction to this chapter, this saved a lot of time and effort in debugging any programs which used the algorithms.

5.3.2 Correlation Using an FFT

The correlation of two data sets can be investigated by comparing both of them directly superimposed, and with one of them shifted left or right. For example the correlation of two functions, $a(t)$ and $b(t)$ say, is a function of $\text{lag}(t)$. The correlation will be large at a value of t where one function leads or lags behind the other function by a value of t . The discrete correlation of two periodic sampled functions can be defined as

$$\text{Corr}(a, b)_j \equiv \sum_{k=0}^{N-1} a_{j+k} b_k \quad 5.3.6$$

It should also be noted that because the following relation holds, the correlation can be made even if the two functions are switched around.

$$\text{Corr}(a, b)(t) = \text{Corr}(b, a)(-t) \quad 5.3.7$$

Now the FFT can be utilised thanks to the discrete correlation theorem which states,

$$\text{Corr}(a, b)_j \Leftrightarrow A_k B_k^* \quad 5.3.8$$

Where A_k and B_k are the discrete FFT's of a_j and b_j and the asterisk indicates complex conjugation. The use of the discrete correlation theorem assumes that certain things are true of the data. The first assumption is that the input signal is periodic. The second is that the duration of the response is assumed to be the same of that the period of the data. Clearly the first requirement can never be met easily using real data, especially when using input signals based upon pseudo random white noise. One solution is to use zero padding. This technique involves adding zero's to the ends of both of the data sets to be correlated so that they satisfy the correlation theorems periodicity requirements. In order to look for a correlation as large as $\pm K$ a buffer of K zero's would be needed to add to the end of each of the data sets. A more detailed treatment of this along with further details can be found in the convolution and correlation sections in "Numerical Recipes in C"².

A good correlation routine can be found in this book, however there are a few important reasons why it has not been used in this project. Firstly it is single precision floating point routine, and secondly it is based around an algorithm that Fourier transforms two sets of data simultaneously, which as we have discussed before is too wasteful. In actual fact most of the work has already been done for us if the data has already been transformed to the

frequency domain. All that is required here is to simply use the routine `drealft` to transform two data sets and then write a small library function to multiply one by the complex conjugate of the other. The function `drealft` can then be used again to inverse FFT the result if required. The structure of the library function written to do this can be seen below.

```
void xcorr(double data1[], double data2[], double output[], unsigned long n)
```

This routine expects to be presented with two data sets in the format that is returned from the FFT routine `drealft`, in this case the arrays `data1[]` and `data2[]`. With the exception of the first two elements of the input arrays (recall that the first and last real values of the Fourier transform are stored in the first two elements of the output array when the routine `drealft` is used) all of the data is stored with the real value first, followed by the complex value. Equation 5.3.8 is used to create a new data set in the array `output[]` by using data from array `data1[]` and `data2[]`. The result from this routine is the spectrum of the correlation. The sign convention of this routine is defined as follows: if `data1` lags `data2`, i.e., is shifted to the right of it, then output will show a peak at positive lags in the time domain. It is important to note that this routine does not inverse FFT that data, it simply returns an array in the same format as the two input arrays were presented. Clearly if an answer in the time domain was required all that one would have to do is add one line of code to inverse FFT the output using the routine `drealft`. The Code for this routine can be found in Appendix B.

5.4 DSP Utility Library

The routines that follow were part of a digital signal processing utility library called Utilz. A collection of routines are presented here that were used to prepare and change the data format. This library was created as most of the routines were used on a very regular basis, which made them

ideal for including in a library. For most of the routines the use is obvious, although a few routines were written specially to overcome problems discovered after taking real data and processing it. The routines can be found in the sub-sections below.

5.4.1 DC Offset Removal (*rmdc*)

As mentioned in the previous chapter, when all of the inputs of the data acquisition system were grounded, there was a small random DC offset that varied each time a data set was taken. For this reason a software routine was written to remove any DC offset present in each data channel. The form of the routine can be seen below.

```
void rmdc(double data[], unsigned long n)
```

This routine takes an average of the input array `data[]` of length `n`, and then subtracts the calculated average from each of the cells in `data[]` and then puts the results back into the array `data[]`. As usual the full listing of the code for this routine can be found in Appendix B.

5.4.2 Apodising Function (*Bartlett*)

A routine was written to apply a Bartlett data windowing function to the data. In effect this routine multiplied the input data by a triangle of height 1 and length `n` which was the same as the input data array length. The structure of the function is shown below.

```
void bartlett(double data[], unsigned long n)
```

Data is presented to the function in the array `data[]` of length `n` and the results are then returned in `data[]`. For the code, see Appendix B.

5.4.3 Phase function (*phase*)

A simple routine was written to calculate the phase of data that had

already been Fourier transformed. The routine is shown below.

```
void phase(double data[], unsigned long n)
```

The input array `data[]`, of length `n`, is expected to be complex, i.e. in the real/imaginary format that is generated by a FFT routine such as `drealfit`. Each pair of real and imaginary values are then used to calculate a phase. The results are then returned in the first half of the input array `data[]`. The code for this routine can be found in appendix B.

5.4.4 Power Function (power)

Another simple function was written to calculate the phase of a data set that had already been Fourier transformed. The structure of the function is shown below.

```
void power(double data[], unsigned long n)
```

Again, just like the phase function in section 5.4.3, this routine took a complex input array `data[]` of length `n` and used each real and imaginary data pair to calculate a power. The results were then returned in the first half of the input array `data[]`. The full code can be found for reference in Appendix B.

5.4.5 Power Averaging Function (Smoother)

This function was used to reduce the size of data sets by averaging lots of small power bins to create fewer larger ones. The structure of the function is shown below. It was expected that all input arrays would contain power data otherwise this routine would be meaningless.

```
void smoother(double data[], unsigned long n, unsigned long no_points)
```


This routine uses an input array `data[]` of length `n` and averages power over each `no_points` in the array. The results are then returned in the first `(no_points)th` of the input array `data[]`. For the program code see Appendix B.

5.4.6 Decibel Scale Function

This routine was written so that power information could be represented in decibels. For this routine it was necessary to define a maximum power to be used as a reference as zero decibels. This subject is covered in greater depth in the the following subsection. The structure of the routine is shown below.

```
void db(double data[], unsigned long n)
```

This routine accepted an input array `data[]` of length `n` and applied the following formula to each cell in the array.

$$New\ Value = 20 * \log_{10} \left(\frac{Old\ Value}{Maximum\ Value} \right)$$

The new values were then returned in the input array `data[]`. As usual the code for this routine has been included for reference in Appendix B. The section that follows describes how the reference values for defining zero decibels were calculated.

5.4.6.1 Zero dB Calibration

It was necessary to define zero decibels so that a reasonable indication could be given of how much of the data acquisition board's dynamic range was being used. As described earlier, all input values were scaled so that the data was in the range of ± 1 . A program was written to synthesize an ideal sine wave data set with a complete number of cycles for data sets of an integer power of 2 long. The ideal sine wave was given maximum and

minimum values of ± 1 and was Fourier transformed using the `drealft` algorithm. The power of the data set was then worked out using the "power" routine. The results as expected showed perfect and ideal results. All of the values except one were exactly zero indicating a delta function had been obtained. The one value that was not zero was an integer exactly half of the number of data values that had been passed to the FFT algorithm. The procedure was also done for input data with other ranges apart from ± 1 . The results obtained can be seen in the table below.

FFT Size	Max & Min Data Values \pm	Maximum Power Value
65536	1	32768
65536	2	65536
65536	4	131072
32768	1	16384
32768	2	32768
etc.	etc.	etc.

The results were used to create the decibel scale function in the previous section. The full program used to generate these results has been included for reference in Appendix B.

5.4.7 Phase Correction Function (*PhaseCrct*)

This routine was written to correct phase data sets that jumped through 2π . Typically with a big cross correlated data set of phase values, the phase could pass through 2π several times. This was due to the fact that the `arctan` command in C could only return values between $\pm\pi$. A simple routine was used to add or subtract 2π to the data values, thus removing discontinuities in the data. The structure of the function is shown below.

```
void stitch(double data[], unsigned long n, unsigned long skip)
```

An input array `data[]` of length `n` is phase corrected and the results are returned inside the input array `data[]`. A number of values at the start of the array, defined by `skip`, are missed by the routine. This was due to expected instability at frequencies very close to DC. The routine worked by looking at the difference between successive values, and because real data was noisy each value tested was compared with an average of the last ten corrected values. The use of this routine is discussed in more detail in the following chapter. The program can be found in Appendix B.

5.4.8 Phase differential Function (diff)

This routine was written to differentiate the corrected phase of a cross correlated data set. The structure of the routine can be seen below.

```
void diff(double data[], unsigned long n)
```

For an input array `data[]` of length `n`, each value was subtracted by its previous value in the array. Because the `x` values of the data were discrete they were disregarded in the routine, and left for the user to add an appropriate scale factor. Results were returned in the input array `data[]`. The need and use of this routine is discussed in the next chapter, whilst the code may be found in Appendix B.

5.4.9 Constrained Regression Function (ConReg)

This routine was written as an alternative to the last function described in section 5.4.8. It was used to perform constrained regression on a phase corrected data set that had been cross correlated. This routine assumed that all gradients on a graph of phase difference against frequency should pass through the origin. In effect this meant that to perform the constrained

regression each value of phase difference was divided by its frequency value. The structure of the routine is shown below.

```
void ConReg(double data[], unsigned long n, unsigned long miss)
```

All values of the input array `data[]` of length `n`, except for those at the beginning defined by `miss`, were processed by the routine. The results were returned in the input array `data[]`. The need and use for this routine are again discussed in the next chapter. The full source code for the routine has been included in Appendix B.

5.5 System Utility Library

This library was constructed to take care of memory allocation and deallocation. One routine was used to allocate the memory, whilst another separate routine was used to deallocate it. Several pairs of routines were written to cover each of the basic types in C such as float, double, int etc. Since the routines for each of the types was similar, only one of the types is presented here. The chosen type here is double, but all of the other types can be found with the appropriate code in Appendix B. The routine used for allocating double precision floating point arrays is shown below.

```
double *dvector(long nl, long nh)
```

This routine was used to allocate named arrays of doubles from `nl` to `nh`. The routine also had an error trapping routine to test whether or not the memory had been successfully allocated. If the memory was unable to be allocated, an error was printed to the screen and then the program was aborted. The error trapping routine along with the code for this structure can be found in Appendix B.

The routine used for deallocating an array of doubles can be seen below. In this case there was no need to check for errors, but great care had to be taken to make sure that the deallocation routine used the same variables as the allocation routine. If the deallocation routine was not use correctly, memory was not freed when the program ended and was lost until the next time the computer was restarted. A typical indication of this happening would be the computer running out of memory after a program had been run several times.

```
void free_dvector(double *v, long nl, long nh)
```

The routine deallocated a named double vector allocated with the previous routine `dvector()`. The full source code can be found in Appendix B.

5.6 Other Libraries

5.6.1 Statistics Library (*stats*)

A statistics library was created for two routines that computed the mean and linear regression of data sets. The structures of the two routines can be seen below.

```
double LRyx(double data[], unsigned long n)
double MeanAv(double data[], int start, int finish)
```

The Linear regression routine (`LRyx`) took an input array `data[]` of length `n` and performed Y on X linear regression. The equation for a straight line was generated and the gradient of it was returned when the routine had finished. The `MeanAv` routine was used to average all or part of the input array `data[]`. The cells between `start` and `finish` were used to calculate a mean. When the routine had finished running the mean was returned. The full source code fro

both of these routines can be found in Appendix B.

5.6.2 Range Library (*Range*)

This library contained a function called *Range*, which was used to find co-ordinates of a target from the path differences between channels. For illustration purposes, only one routine is discussed here, and that is the *Range* function for a three port system. Changing this routine to work with more than 3 ports or channels was a relatively simple task and it has been assumed that the reader is capable of this. The structure of the routine is shown below.

```
void Range(double m, double n)
```

The routine took input variables, *m* and *n*, calculated co-ordinates for a target and printed the results to the screen. The variables *m* and *n* were the path differences between the centre port and both of the side ports in the three port system. This has already been covered in Chapter 3 in section 3.4 and can be referred to for reference. As usual the source code for this routine can be found in Appendix B.

5.7 Testing of Library Functions and Routines

Most routines were quite easy to test and this was done as the routines were written. As previously mentioned this was not generally true about the routines in the digital signal processing library - see section 5.3. For this reason a program was written to create synthesized data to test the FFT and correlation routines. The program was similar to the one described in section 5.4.6.1 that was used to define a value for zero decibels. An ideal sine wave was generated and then fast Fourier transformed to check that its transform was what was expected. As expected a perfect delta function was obtained. The cross-correlation routine was tested by taking the synthesized

sine wave data and cross-correlating it with itself, thus in effect performing an autocorrelation. The data that was used for the autocorrelation and the results that were obtained can be seen in figure 5.7 below.

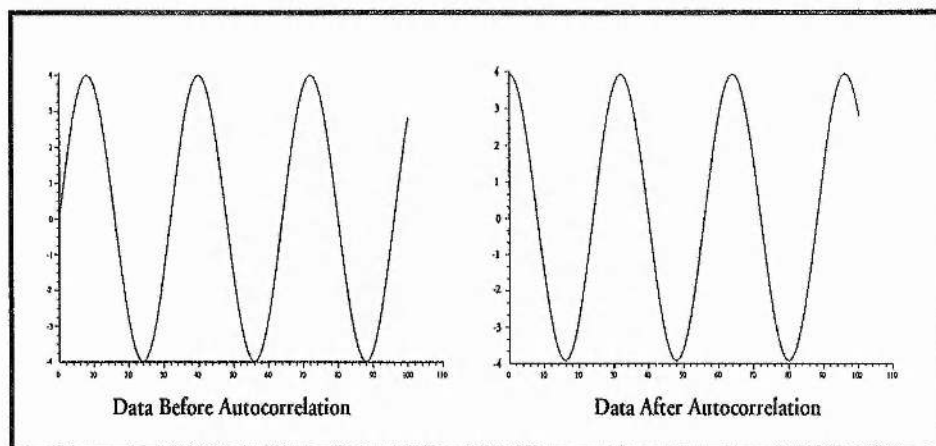


Figure 5.2 Testing the Cross-Correlation Routine

The test process involved using the FFT routine `drealft` to transform the data, the `xcorr` routine to perform the cross correlation (in this case an autocorrelation), and finally the `drealft` routine again to inverse transform the resultant data. It can be seen, as expected, in figure 5.7 that the result of autocorrelating the sinewave data was a cosine wave. This provided a very clear indication that the Fourier transform and cross-correlation software routines were working properly.

5.8 An Example: A four Channel Analyser

A four channel spectrum analyser program was constructed from the routines that have been described in the previous sections. This program proved to be a very useful means of checking what was happening on each of the input channels and provided an indication of how much of the data acquisition systems dynamic range was being used. This program took up to eight input channels and calculated the spectrum for each of them. After the

data had been loaded using the `Load_Data` command, each channel was Fourier transformed using the `drealft` routine. The power was then calculated using the `power` command, and then the data was changed to a decibels scale with the `db` command. Output files were then saved to disc using the `Ram_write` command.

Constructing the four channel spectrum analyser program was a very trivial task because most of the work had already been done already by using the commands from the library functions. The main program was constructed in only a few minutes and was very small in size. The program can be found in Appendix B for reference. This program was used to take data for testing the cross talk in the data acquisition system's control box. The spectrum that was obtained can be seen in chapter 4, section 4.9.

Another important use of the routines was to create a program that emulated the Stanford spectrum analyser that had been used to take the data in early experiments. The four channel analyser that has been described above was used to perform large 32k or 64k transforms on the input data, while the Stanford analyser performed lots of 1k Fourier transforms and averaged the power giving a very stable results. A routine called `StanEmul` was created to perform lots of small FFT's and average the power from each of them. The program can be found in Appendix B for reference. Once the program had been written, it was also made into a library function so that it could be included as just one line of code in a main program.

5.9 Conclusion and Comments

All of the routines that have been discussed were created on an Acorn RiscPc using Acorn C/C++. A few years ago, computing power and price would have meant that the signal processing used in this Ph.D. study would have been prohibitive. By using the Acorn platform many technical difficulties were overcome that would have been very difficult to solve on any

other platform. It was unfortunate that a suitable data acquisition board could not be found for the RiscPc. It is understood at the time of writing this thesis, that a suitable data acquisition board is being designed for the RiscPc.

All of the routines worked very satisfactorily, and were computed relatively quickly. The RiscPc was fitted with a Digital Strong Arm processor running at 200Mhz and was as fast as a Dec Alpha workstation, however it would have been better if some of the programs could have run faster. A program such as the four channel analyser took approximately 40 seconds to run and was one of the less computational intensive programs. This said, before the Strong Arm processor was purchased, the same routine was taking around 15 minutes! The programs were also memory intensive and accordingly the RiscPc was fitted with 50 Megabytes, taking full advantage of the recent memory price crash.

It is expected that a multi processor board will be released for the Acorn platform in the future that can be populated with several Strong Arm processors. The digital signal processing routines would have been an ideal application for such a board, as the data for each of the channels could have been computed simultaneously. The routines could also benefit in the future if a version of the Strong Arm is released with a floating point unit. Overall the future for this type of signal processing looks very bright.

References

- ¹ Al Kelley & Ira Pohl, *A Book on C Programming in C*, The Benjamin Cummings Publishing Company Inc, Third Edition, 1995.
- ² *Numerical Recipes in C - The Art of Scientific Computing*, W.H.Press S.A.Teukolski W.T,Vetterling & B.P.Flannery, Cambridge University Press, Second Edition 1994.

CHAPTER 6

Experiments and Results

6.0 Introduction

In this section the results from various different types of three port experiments are presented. The first experiment that is presented in section 6.1 was performed with the test loudspeaker in the upright position. When the loudspeaker is in an upright position its woofer and tweeter are situated in a line which is perpendicular to the receiving plane. The model described in chapter 3, was used to simulate this situation and it was found that the two sources were indistinguishable. It was only when the loudspeaker was placed on its side that the two sources could be separately resolved. In the first experiments that were performed, the upright position was used because it was felt that in the first instance it might be easier to locate what appeared to be a single source. The system was then extended to five ports and data was then taken for different source position and orientations. When five ports were used, a total of four different linear three port systems could be constructed to process the results. When this was done, the origin of each separate three port system was specified because the centre port was always taken to be the origin. In order that the accuracy of results could be verified, the actual port to target distances were measured by using pieces of string. These measurements were difficult to make in the anechoic chamber, and were only considered to be accurate to within $\pm 2\text{cm}$. Matters were complicated further because the loudspeaker could not really be considered as a point source. Some of the best results presented in this chapter had been used to locate the position of the loudspeaker using the digital signal processing and ranging software. The target co-ordinates that were generated were far more accurate than those that were measured "by hand". The results are now presented.

6.1 First Three Port Experiment

Three of the larger microphones were placed in the anechoic chamber on an optical rail with initial linear spacings of 40cm. A Spondor LS3/5A loudspeaker was placed approximately 175cm opposite the centre microphone on the optical rail and was used as the test source. The equipment was setup as discussed in section 4.2 and shown in figure 4.1. Care was taken in setting the loudspeaker volume and the microphone preamplifier volume to avoid saturation. An oscilloscope and spectrum analyser were connected to the control box's monitor outputs so that the signal levels on each channel could be appropriately adjusted, thus ensuring that each channel was providing a sensible voltage and was not going to be clipped by the data acquisition system. Once the experiment was setup, results were taken by the data acquisition system. A sampling frequency of 52kHz was used to take 102400 samples per channel. A fourth channel was used to record the signal that was being transmitted by the loudspeaker, whilst the other four channels were set to earth. The experiment was then repeated by moving the loudspeaker controlled amounts. For each position, three data sets were taken. After the data had been taken, it was transferred to the RiscPC for signal processing. Each channel was checked for any DC offset using the "rmde" routine and any that was found was removed automatically. Then a FFT of length 65536 (64k) was performed on each channel using the "drealft" routine. The set of spectra that was obtained for each channel can be seen below in figure 6.1.

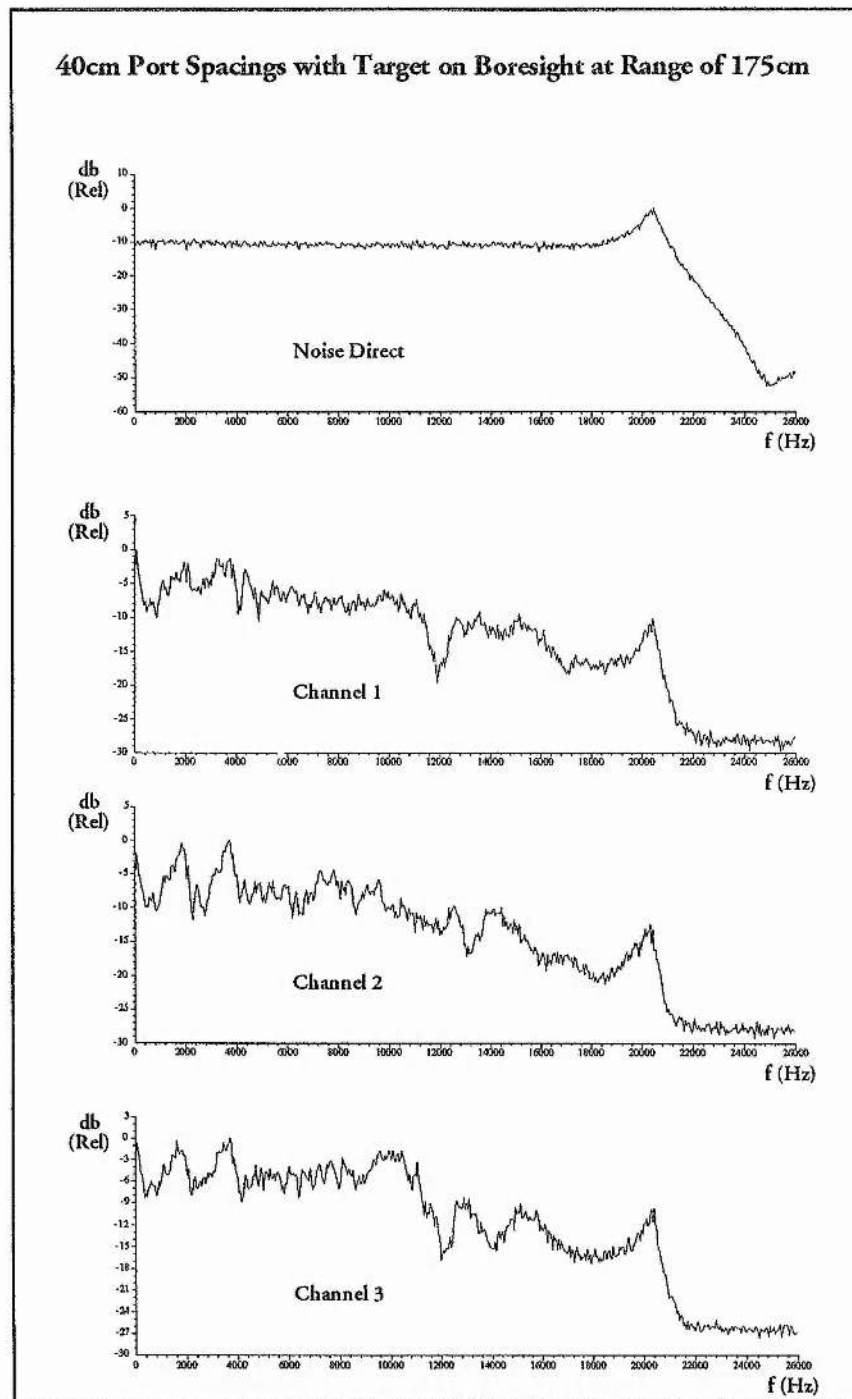


Figure 6.1: Spectra obtained using the 4 channel analyser program.

The graphs that are shown in figure 6.1 were obtained using the 4 channel spectrum analyser program. The top graph shows the spectrum of the input noise source, whilst the graphs underneath show the spectra of the signals received by the three ports. It was noticed that the noise spectrum had an

unusual bump at 21kHz just before the cut off point. This problem was traced to a pilot filter that was used in the white noise box to perform low pass filtering. The pilot filter had drifted and its impedance had changed, thus causing the bump to be seen. The matter was quickly rectified by fitting a new filter. This is a particularly good example that demonstrates the value of recording the input signal, and using the 4 channel spectrum analyser program. Generally the spectrum analyser program was an invaluable aid in troubleshooting the system and for performing quick checks on the input channels. After looking at the spectra of data that had been received by the three microphones, it was quickly realised that there was not a lot of point using a sampling rate as high as 52 kHz. It can be seen in the spectra obtained that the response was not particularly good after 12kHz. Accordingly the decision was made to reduce the sampling rate to 24kHz before proceeding any further.

The data acquisition program was modified and recompiled for the new sampling frequency of 24kHz. The low pass filtering required to comply with the Nyquist criterion was automatically adjusted by the data acquisition board. This was quite a useful feature and was made possible by the boards oversampling. Data was then taken under the same conditions as before. After the spectra had been obtained, the resultant data sets were passed to the cross-correlation routine "xcorr". For this three port experiment two cross-correlations were made. Channel 1 was cross-correlated with channel 2, and channel 2 was cross-correlated with channel 3. Two resultant complex data sets were obtained and were then changed into phase form using the phase routine. The cross correlated phase could then be plotted against frequency as shown in figure 6.2 below.

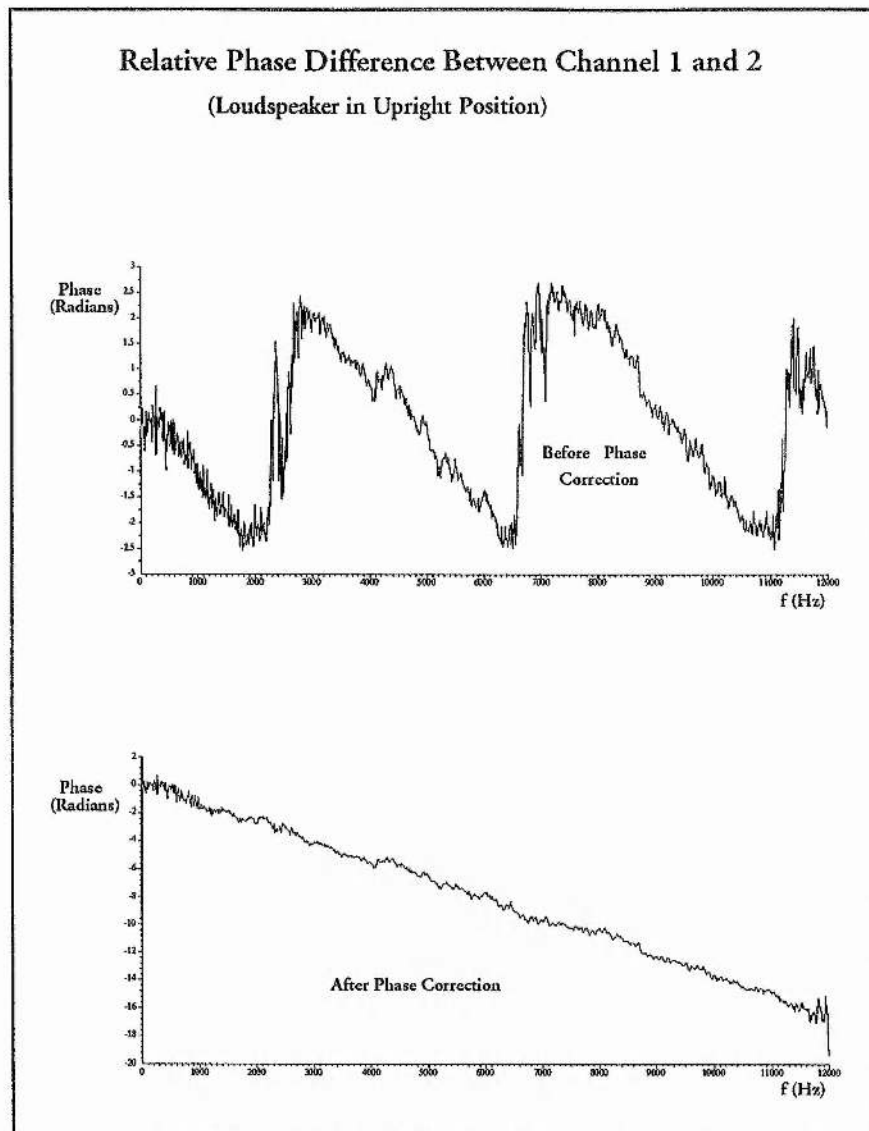


Figure 6.2: Cross-correlated phase between port 1 and port 2.

It can be seen in the top graph of figure 6.2 that the phase passes through 2π at several different points in the cross-correlation. As discussed in the modelling section of chapter 3, the locations of the jumps are dependent on the port spacings and the bearing location of the target. Here the aim was to measure the gradient of the graph so that it could be used to calculate the path difference between ports. In order that the gradient could be found automatically it was necessary to correct for the phase jumps of 2π by using the "phasecrct" routine. If this was not used, the fact that the jumps were taking place at unknown frequencies would have meant that the gradient

would have to of been calculated in different places for each data set. This would have also meant that only a limited amount of the data could have been used. The data shown in the top graph of figure 6.2 was processed by the "phasecrct" routine and the results that were obtained can be seen in the lower graph in figure 6.2. Once the data had been phase corrected, the y on x linear regression routine "LRyx" was used to obtain the gradient of the graph. The same procedure was then used to find a gradient for the corrected cross-correlated phase data from ports 2 and 3. The two gradients were then passed to the "range" routine which was used to calculate target co-ordinates. Three sets of data were taken for each position of the source loudspeaker and the results obtained can be seen in the table below.

Source Displacement	x co-ordinate (cm)	z co-ordinate (cm)
0 cm	2.814	177.03
	2.766	175.90
	2.901	177.64
-5cm	-2.513	176.64
	-2.508	178.34
	-2.550	177.44
-10cm	-7.416	175.95
	-7.499	176.89
	-7.442	177.60
-15cm	-12.708	175.71
	-12.627	175.38
	-12.616	174.86

Table 6.1.1: Target co-ordinates using 40cm port spacings.

The results shown in table 6.1.1 are very good and were consistent for each of the three data sets taken for every position of the source. The results also changed in the expected manner when the source was moved in 5cm increments. For all of the results, 11kHz out of the 12kHz bandwidth was used by the "LRyx" linear regression routine. In the lower graph of figure 6.2 it can be seen that graph of relative phase against frequency is not a perfectly straight line such as the ones that were obtained when using data generated by

the model. This was probably due to noise in the microphones and phase anomalies in the source loudspeaker. Another reason is that in reality the loudspeaker cannot be considered as a point source, although it was expected that this would have caused smaller fluctuations than those observed. Because of the noise that was generated, it was necessary to perform the regression on almost all of the data to get accurate results.

6.2 Three Port Experiment With The loudspeaker Sideways

Since good results had been obtained in the previous section with the source loudspeaker in the upright position, it was decided to turn the loudspeaker on its side to see if the two separate sources could be easily detected. This experiment did not work particularly well, although it could be seen that there were two sources present instead of just one. The data processing proceeded in the same way as for the experiments in the previous section. The results obtained for the relative phase between port 1 and 2 were plotted against frequency and can be seen below in figure 6.3.

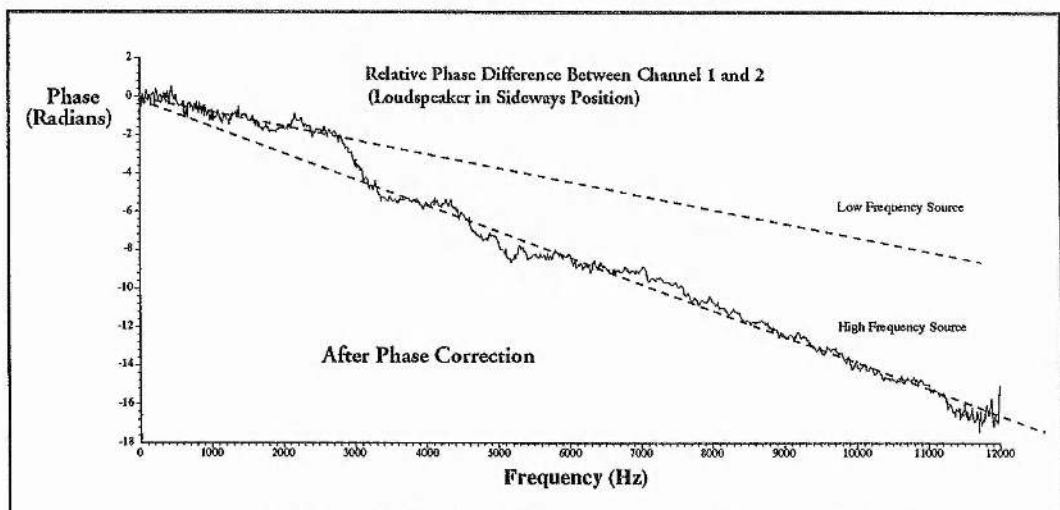


Figure 6.3: Cross-correlated phase between port 1 and port 2.

Two separate gradients can be made out in the diagram that represent the

loudspeaker's woofer and tweeter, and these are shown by the two dotted lines. The "LRyx" linear regression routine was then applied to two different parts of the data which corresponded to the two gradients. The "range" routine was then used to calculate co-ordinates for the woofer and the tweeter. The results that were obtained were inconsistent and often meaningless given the size of the anechoic chamber. Often just by changing the area that the regression was performed on by a very small amount had drastic effects on the results. Basically because the regression was being performed on a smaller bandwidth of the data, a bigger percentage error was obtained in the gradient. This was considered a resolution problem as the errors obtained were far too big to account for the loudspeaker not behaving as two point sources. In effect this meant that there was not enough resolution to image the two sources properly. This topic is discussed again with greater detail in this chapter in section 6.6.

6.3 Expansion to Five Ports

Having completed the previously described experiments with three ports, it was decided to extend the system for use with five ports before proceeding any further. Although the three port system had not been properly characterised yet, this could still be achieved whilst using five ports and at the same time gave some useful advantages. A five port system can in effect be considered as a number of separate three port systems. Let us suppose that we have a linearly spaced five port system with ports that we will call port 1 through to port 5. If we then try to break down this geometry into sets of linearly spaced three port systems, we can obtain four separate systems. The first three are relatively obvious: port 1 - port 3, port 2 - port 4, and port 3 - port 5. The remaining three port system has linear port spacings that are twice as big as the other systems and consists of port 1, port 3, and port 5. In this kind of arrangement and by using five ports, a lot more information can be obtained from each set of experimental data. This is not the only use for a five

port system. The five ports can also be used to form a two dimensional cross. In effect this is two sets of three ports perpendicular to each other that both share the same centre port. This combination of three port systems in two different dimensions means that the location of a target can be calculated to be at a certain place in a hemisphere instead of a semicircle.

The results that are presented in the following sections were all calculated from data sets that were generated on a five port system. The results were then processed for different sets of three port systems. Many different data sets were then taken for a variety of different port spacings. The target loudspeaker was used in both horizontal and vertical orientations, and was also moved by controlled amounts. The experiments are now presented.

6.4 Five Ports With Variable Port Spacing

The purpose of these experiments was to see in reality how much the performance of a three port system depended on the port spacing. A five port linear array was setup on the optical rail using the larger microphones. The target, or source, which was a Spondor LS3 5A monitor loudspeaker in an upright position, was used to play white noise from the home made pseudo random white noise generator. A sampling rate of 24kHz was then used to make a series of measurements for 20cm, 30cm, and 40cm port spacings.

6.4.1 20cm Port Spacings

Five microphones were placed on the optical rail at positions marked on the scale of 90, 110, 130, 150, and 170cm, and were labelled port 5 through to port 1 respectively. The Spondor LS3 5A loudspeaker was positioned at a range of approximately 160cm opposite the 125cm mark on the optical rail. For each experiment 5 data sets were taken to show the extent

of random errors. The first results were calculated from a three port system comprising of ports 1, 2, and 3. For a three port system the centre port is always taken as the origin, and in this case was port 2 at a position of 150cm. Positive x values that were calculated represented a displacement to the right when looking at the target from the origin port - i.e. the target would be situated opposite the optical rail at a position calculated by subtracting the x value from the origin port value. The results obtained can be seen in the table below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	24.91	173.27
2	25.01	174.03
3	24.99	173.66
4	24.98	174.01
5	24.88	173.29

Table 6.4.1: Results from ports 1-3 with 20cm spacings (150cm origin)

The calculations were then repeated for the same data sets, but this time ports 2, 3, and 4 were used instead. This gave a new system origin of 130cm. The results can be seen in table 6.4.2 below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	2.18	141.35
2	2.13	140.52
3	2.17	140.88
4	2.16	141.07
5	2.17	141.69

Table 6.4.2: Results from ports 2-4 with 20cm spacings (130cm origin)

The calculations were then performed for ports 3, 4, and 5 with a new origin of 110cm. The results are shown in table 6.4.3 below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	-22.04	217.86
2	-22.18	217.93
3	-22.23	218.10
4	-22.01	217.78
5	-22.17	217.90

Table 6.4.3: Results from ports 3-5 with 20cm spacings (110cm origin)

The final calculation for these data sets was performed using ports 1, 3, and 5 giving a new port spacing of 40cm and an origin of 130cm. The results can be seen below in figure 6.4.4.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	3.42	162.87
2	3.42	162.74
3	3.44	162.97
4	3.41	162.75
5	3.42	163.04

Table 6.4.4: Results from ports 1, 3, 5 with 40cm spacings (130cm origin)

Generally the overall results here are poor. Although each different three port system gave consistent results for each data set, they were not in agreement for the general location of the target. The most promising results were obtained using the data from ports 1, 3, and 5. This was probably due to the increase in port spacing from 20cm to 40cm.

6.4.2 30cm Port Spacings

The microphones were then repositioned on the optical rail at positions of 70, 100, 130, 160 and 190cm. Again they were labelled port 5 down to port 1 respectively. The first results were calculated from ports 1, 2 and 3, with an origin of 160cm. The results are shown below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	33.65	163.15
2	33.65	163.21
3	33.64	163.32
4	33.62	162.95
5	33.79	163.53

Table 6.4.5: Results from ports 1-3 with 30cm spacings (160cm origin)

Another set of calculations were then performed using ports 2, 3, and 4. The origin was redefined as 130cm. The results are shown below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	3.47	161.62
2	3.45	161.39
3	3.47	161.83
4	3.47	161.65
5	3.48	160.78

Table 6.4.6: Results from ports 2-4 with 30cm spacings (130cm origin)

The next calculations were made using ports 3, 4, and 5. The origin was now defined as 100cm. The results are shown below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	-26.96	167.68
2	-27.02	168.04
3	-26.92	167.45
4	-26.96	167.78
5	-27.00	167.56

Table 6.4.7: Results from ports 3-5 with 30cm spacings (100cm origin)

The final calculations were made using ports 1, 3, and 5. This gave a new port spacing of 60cm and defined the origin as 130cm. The results are shown below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	3.72	163.37
2	3.71	163.34
3	3.70	163.62
4	3.69	163.40
5	3.70	163.25

Table 6.4.8: Results from ports 1, 3, 5 with 60cm spacings (130cm origin)

It was very noticeable that the results obtained were far better than the previous results that used port spacings of 20cm. All of the z co-ordinates were within a range of ± 4 cm and the x co-ordinates were within ± 1 cm.

6.4.3 40cm Port Spacings

For the last experiment in this series, the microphones were repositioned on the optical rail to 30, 70, 110, 150, and 190cm. As usual, they were labelled port 5 down to port 1 respectively. The source loudspeaker was situated approximately 160cm opposite the 127cm mark on the optical rail. The results calculated using ports 1, 2, and 3, can be seen below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	22.97	155.89
2	23.02	156.17
3	23.01	156.12
4	23.00	156.12
5	22.99	156.23

Table 6.4.9: Results from ports 1-3 with 40cm spacings (150cm origin)

As before, new values were then calculated using data from ports 2, 3, and 4. The origin of the system was then redefined to be 110cm. The results are shown below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	-16.88	164.05
2	-16.83	163.63
3	-16.85	163.64
4	-16.85	164.04
5	-16.87	163.85

Table 6.4.10: Results from ports 2-4 with 40cm spacings (110cm origin)

The next set of results was calculated using the data from ports 3, 4, and 5. The origin was defined as 70cm. The results can be seen below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	-57.58	167.21
2	-57.65	167.33
3	-57.61	167.11
4	-57.51	167.05
5	-57.69	167.56

Table 6.4.11: Results from ports 3-5 with 40cm spacings (70cm origin)

Finally, results were calculated using ports 1, 3, and 5, which meant that the port spacing changed from 40cm to 80cm. The origin was now at 110cm, and the results obtained can be seen below.

Data Set No	x Co-ordinate (cm)	z Co-ordinate (cm)
1	-16.09	162.70
2	-16.09	162.60
3	-16.10	162.54
4	-16.09	162.68
5	-16.12	162.81

Table 6.4.12: Results from ports 1, 3, 5 with 80cm spacings (110cm origin)

With the exception of the results shown in table 6.4.9 that were taken with ports 1, 2, and 3, the rest of the results were very good, especially those in table 6.4.12 that were generated by a port spacing of 80cm.

6.4.4 General Comments

After all of the results had been taken and processed it was soon realised that the best results were obtained when using larger port spacings. This was expected and in effect meant that with port spacings of 20cm, there was not enough resolution to range the target properly. As the port spacing was increased, there was an increase in resolution and this is reflected in the results. It is worth mentioning that all of the results in this section were calculated using the y on x linear regression routine over a bandwidth of approximately 10.2kHz. The bandwidth that is used for the regression also effects the resolution. This subject is covered in greater detail in section 6.7 of this chapter.

One of the results obtained using 40cm port spacings was unexpected and required further investigation. The results shown in table 6.4.9 indicated an error in the z co-ordinate that was bigger than expected. Furthermore, this error systematically appeared in all five of the results calculated. This problem had been noticed before in other experiments and was eventually explained by taking a step back and looking at the three port geometry again. Some more error measurements were made by introducing errors into the path differences between ports, and then calculating the range to see how big an error was produced. It was quickly realised that a small percentage error in the distance value between ports could generate a bigger percentage error when calculating the x and z co-ordinates. Graphs of these errors can be found in Appendix A for reference purposes. After looking at the position of the source that was used in the 40cm port spacing experiment, it was discovered that it was situated opposite a position on the optical rail approximately half way between port 2 and port 3. When a source is in this position it means that there is hardly any path difference between signals arriving at port 2 and port 3. As we have already seen, gradients from graphs of relative phase against frequency are used in experiments to calculate the

path differences between ports. In this situation the gradient of such a graph would be very close to zero and a lot of system generated noise would be seen. The gradient of the line in this case is small and is hidden inside the noise. It therefore follows that there is a large percentage error in the gradient which then becomes a large percentage error in the path difference between ports. In effect we can say that there is a decrease in resolution when a target is situated opposite the midpoint between two ports. These errors arise because of the way the new three port geometry works. They were also found to decrease with increasing port spacing.

There were other interesting features of the data that were given more consideration. Each data set that had been taken using five ports was used in four different three port systems to find the location of a target. For each different three port system, slightly different co-ordinates were calculated for the target. It was decided to investigate this further by moving the target by controlled amounts to see if the different co-ordinates all moved by the same amount. This is dealt with in the following section.

6.5 Five Ports With Variable Target Co-ordinates

The five microphones were setup on the optical rail in a linear array with spacings of 40cm at positions of 30, 70, 110, 150, and 190cm. As before they were labelled port 5 down to port 1 respectively. In the experiments in this section, the loudspeaker was in the upright position and was moved an increment of 5cm between two sets of four readings. The source was imaged using every possible linear three port system that could be made from the five ports, and the results were compared. More measurements were then made by incrementing the loudspeaker position by further amounts of 5cm. The source loudspeaker was moved in a parallel plane approximately 160cm from the receiving plane. The first set of results were taken with the loudspeaker approximately opposite the 100cm mark on the optical rail. After four data

sets had been taken the loudspeaker was moved so that it was opposite the 105cm mark and more data sets were taken. The co-ordinate values that were obtained are now presented and can be seen in table 6.51 below.

Source Displacement	Ports Used	System Origin	x co-ordinate (cm)	z co-ordinate (cm)	Port Spacing
0cm	1,2,3	150cm	50.10 50.19 50.23 50.12	155.04 155.27 155.16 154.99	40cm
-5cm	1,2,3	150cm	45.35 45.31 45.34 45.35	154.49 154.54 154.48 154.52	40cm
0cm	2,3,4	110cm	11.65 11.63 11.63 11.65	163.11 162.79 162.47 162.96	40cm
-5cm	2,3,4	110cm	6.71 6.68 6.73 6.70	162.89 163.06 163.06 162.90	40cm
0cm	3,4,5	70cm	-28.38 -28.41 -28.43 -28.39	163.66 163.62 163.75 163.75	40cm
-5cm	3,4,5	70cm	-33.37 -33.39 -33.32 -33.37	163.93 163.97 163.62 163.83	40cm
0cm	1,3,5	110cm	12.02 11.99 12.01 12.03	161.30 161.17 161.08 161.17	80cm
-5cm	1,3,5	110cm	7.17 7.17 7.17 7.17	161.10 161.14 161.17 161.10	80cm

Table 6.5.1: Results for different three port systems.

Upon inspection of the data, it can be seen that although different results were obtained for each three port system, when the target was moved, all of the different systems detected a movement of the same amount. This was quite reassuring, and indicated that the system was working properly. The differences in co-ordinates obtained from the different three port systems was probably dependent on how accurately the path differences between ports could be measured (dependent on the position of the target). Another reason for the differences could have been that the different systems were seeing different images because the loudspeaker was not behaving as a point source. The results obtained for the 80cm port spacings using ports 1,3,and 5, were very impressive and some more measurements were made with the loudspeaker in different positions. Here the loudspeaker was approximately 160cm opposite the 100cm mark on the optical rail and the system origin was defined to be 110cm. These results are shown in table 6.5.2 below.

Source Displacement	x co-ordinate (cm)	z co-ordinate (cm)
0cm	12.02	161.30
	11.99	161.17
	12.01	161.08
	12.03	161.17
-5cm	7.17	161.10
	7.17	161.14
	7.17	161.17
	7.17	161.10
-10cm	2.33	161.46
	2.35	161.25
	2.33	161.32
	2.34	161.28
-15cm	-2.61	161.32
	-2.59	161.21
	-2.61	161.33
	-2.59	161.25

Table 6.5.2: Results from calculations using 80cm port spacings.

These results were excellent, in fact it had become clear that when 80cm port

spacings were used, the system could find the location of the loudspeaker in the anechoic chamber far more accurately than could be measured "by hand"!

6.6 Five Ports With Sideways Orientated Source

Here the setup was exactly the same as in the previous section apart from the fact that the loudspeaker was turned on to its side. The loudspeaker's x co-ordinate was moved by increments of 5cm, and again five data sets were taken in every position. Upon inspection of the co-ordinate values that were calculated from the data sets, it was realised that the only good results were produced by one out of the four three port systems. This was the system that used ports 1, 3, and 5, and had a port spacing of 80cm. This was not really surprising because the previous results that had been obtained in section 6.2 used 40cm port spacings and also did not give good results. The results obtained using the 80cm port spacings are shown below in table 6.61. The loudspeaker was situated in a plane approximately 160cm parallel to the receiving plane. The origin was defined to be at the 110cm mark.

It can be instantly seen that the results obtained using the 80cm port spacings were very good. It was realised that by increasing the port spacing up to 80cm, an increase in resolution had been obtained, thus allowing both sources to be resolved more accurately. It was also noticed that the tweeter was resolved more accurately than the woofer. For all of the results calculated here, the linear regression routine "LRyx" used a frequency bandwidth of 2kHz to calculate the path differences between ports for the woofer. Similarly a bandwidth of 5.5kHz was used for the tweeter. Since more data was used to calculate the co-ordinates for the tweeter, it explained why a greater accuracy was observed. It was quite interesting to see that throughout the results shown in table 6.61, the woofer was detected to be approximately 4cm further away

than the tweeter. This was explained by the fact that the centre of the woofer's cone was physically about 4cm further back in the box of the loudspeaker than the tweeter.

Source Displacement	Woofer		Tweeter	
	x co-ord	z co-ord	x co-ord	z co-ord
0cm	11.53	165.34	-3.02	161.27
	11.83	165.30	-3.02	161.22
	11.99	166.84	-3.02	161.23
	11.82	165.03	-3.03	161.13
	11.99	166.41	-3.02	161.27
5cm	16.29	163.77	2.21	161.45
	16.57	163.66	2.19	161.51
	16.45	164.51	2.18	161.58
	16.43	164.47	2.18	161.61
	16.24	164.78	2.21	161.46
10cm	21.84	164.74	7.43	161.81
	21.64	163.58	7.42	161.68
	22.12	163.93	7.43	161.75
	21.49	163.79	7.44	161.74
	21.69	165.07	7.43	161.68
15cm	26.94	166.32	12.28	161.98
	26.80	165.03	12.29	161.88
	26.89	165.80	12.31	161.96
	26.64	163.68	12.30	161.98
	26.88	164.28	12.28	161.98

Table 6.6.1: Results for a sideways loudspeaker with 80cm port spacings.

There were some complications in performing these calculations that should be explained. Sometimes the jump that occurred at the woofer-tweeter cross-over frequency also coincided with a phase jump of 2π that was caused by the limited field of view. Such an event can be viewed in the top graph of figure 6.4. When the phase correction routine "phasecrct" was used, it was unable to detect the combination of events and did not work in the way that it was originally intended to. The results that were obtained after using the "phasecrct" routine are shown in the lower graph of figure 6.4.

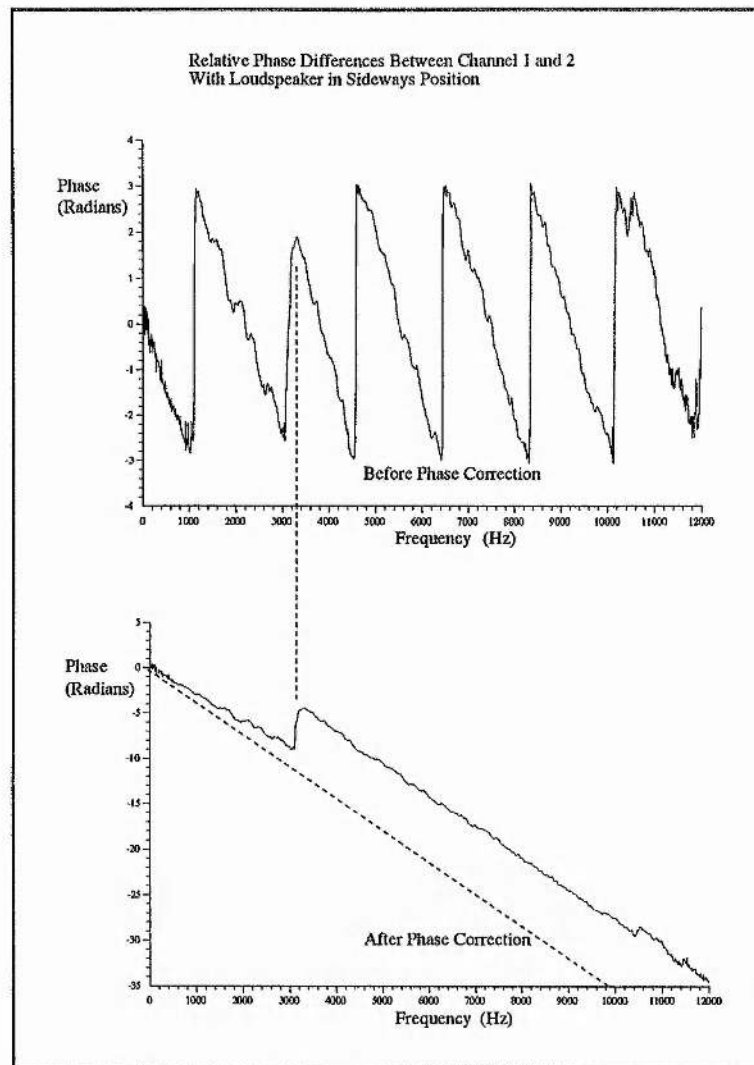


Figure 6.4: Cross-correlated phase against frequency for 80cm port spacings.

The line which represents the tweeter has been offset by the "phasetct" routine and can no longer be extrapolated to the origin. A dotted line can be seen in the lower graph of figure 6.4, and this represents where the tweeter data should be. Although this was undesirable, this effect did not mar the results in any way as only the gradients of the lines were needed to perform the ranging calculations. It was decided not to spend any time to try and correct this problem.

The lower graph in figure 6.4 can be compared with the similar graph in figure 6.3 where port spacings of 40cm were used. It was

immediately apparent that the data obtained with the 80cm spacings was far smoother, and hence far easier to perform linear regression on, than the data obtained with the 40cm spacings.

6.7 Comments and Observations

The results that were generated by the various experiments that have been described in this chapter were generally very good and allowed some very important observations to be made. The results that were presented in section 6.4 showed that an increase in resolution was obtained by increasing the port spacing. This was expected and had already been predicted theoretically in chapter 3, section 3.1. The results in section 6.4 also showed that the system was less accurate when the path difference to the target for two adjacent ports was close to zero - i.e. the target was situated at a distance perpendicular to the midpoint between two adjacent ports. This accuracy problem was caused by the way the three port geometry had been solved, and was more of a processing problem than an experimental problem. It was found that the effect was minimised when a larger port spacing was used. This means that we can say that there is a slight decrease in resolution when the path difference between two adjacent ports is very small. It was also shown that different three port systems returned different target co-ordinates.

In section 6.5, a large port spacing was used and the target was moved into various different positions. Although in each position different sets of three ports gave different target co-ordinates, when the loudspeaker was moved by a set amount, all of the different systems were in agreement in the amount that the target had moved. The differences obtained by the different systems was put down to differing accuracies in the values for path differences between adjacent ports, microphone noise, and errors in microphone positions caused by their finite size. Again all of these effects were minimised by increasing the port spacing. Another set of results were

shown in this section for different displacements of the loudspeaker using 80cm port spacings. As expected, even better results were obtained than those obtained with the the smaller port spacings in section 6.4.

When the loudspeaker was turned on its side, the three port system could then see two separate sources. This was first tried in section 6.2 for a port spacing of 40cm. Whilst this port spacing was enough to range an upright loudspeaker, it did not give good results when the loudspeaker was turned on its side. In section 6.6, the experiment was repeated with a port spacing of 80cm. This port spacing gave enough resolution for the two sources to be distinguished and gave very accurate results. It was noted that the results were still not as accurate as those obtained with the same port spacing for the loudspeaker in the upright position. The basic difference between these two experiments was the amount of the available data bandwidth that was used to calculate target co-ordinates. When a loudspeaker was in a sideways position, smaller data bandwidths were used to calculate the target co-ordinates than when the loudspeaker was in the upright position. In effect, for a greater bandwidth of data, an increase in resolution and accuracy was obtained.

The findings from the various experiments have shown that best results were obtained by using the largest port spacings possible. This is particularly important if the loudspeaker is on its side and more than one source is detected by the system. Once data had been taken, the best results were calculated using as much of the data as possible. It has also been revealed that the accuracy of the system is dependent upon the path difference between two adjacent ports and the target.

CHAPTER 7

A Final Review

7.0 Conclusion

This thesis has shown the design and successful implementation of a near field passive ranging system for the audio frequency band. The work presented, spans system design all the way through to experimental use of a working system. This study started from the very beginning with no equipment and a shortage of money. Accordingly, the equipment that was designed, built and tested has also been presented. This was discussed in greater detail in chapter 1.

In chapter 2, a basic introduction was given to the various components and properties that make up a three/multi port system. Consideration was given to the properties of the propagation of sound in air, and loudspeakers and microphones were presented as transmitters and receivers for the medium. The theory for white noise sources was then presented along with an introduction to frequency analysis techniques. Finally the use of the concepts were demonstrated in some early experiments, and the results were used to provide a new direction for the overall approach to this project.

An introduction to three port geometry was presented in chapter 3. The concepts of field of view and resolution were introduced, and it was shown that their conflicting requirement usually results in a trade-off. A three port interferometer, originally designed for use with mm waves, was introduced that was capable of passively ranging targets under certain conditions in the mid-far field. It was then shown that by using a new geometry design, a three port system could be made to passively range in the

near field without making any approximations. A model was then constructed to test and characterize the new three port theory.

Chapter 4 saw the design and construction of all of the elements that were needed to build a new three/multi port interferometry system. The design requirements we presented along with an overview of the new system and a description of the chosen parts. Each of the components was then presented in full detail and full construction details were given. Finally, the whole system was put together, characterised and tested.

In Chapter 5, the digital signal processing routines that were used for processing data obtained with the new three port system were presented. Choice of hardware platform and programming language were discussed, and as a result all of the data processing was carried out on a RiscPC using the Acorn C/C++ programming language. The digital signal processing algorithms were then presented, and it was shown that they had also been incorporated into library functions for ease of use. The algorithms were tested and then demonstrated in some example programs. The routines in this section were catalogued in Appendix B for reference purposes.

After the design of the new system and when the digital signal processing routines had been completed, the two parts were put together and various experiments were performed. The experiments and results were shown in chapter 6. Various types of three port system were used, and it was shown the resolution obtained was dependent on the amount of available data and the port separation. The results that were obtained were used to fully characterise the working system. In general, the results were very good and fitted in very well with theoretical results that had been generated by the model that was described in chapter 3.

7.1 Future Work

It had been shown in chapter 6 that targets could be located with greater accuracy when large port spacings were used, and when large bandwidths of data were used for performing the ranging calculations. The loudspeaker that was used as a source in the experiments basically comprised of two reasonably wide band sources which had made it relatively easy to range. If a source was used that had a narrower bandwidth, it would have been consequently harder to range. In effect a narrower bandwidth means that there is less available data that can be used to find a range for the source. One possible way to overcome this problem, might be to increase the length of time over which the data is taken. If all of the extra data was transformed into the frequency domain, it would mean that the resulting frequency bins would be smaller, and hence more data would be available for ranging and give an increase in resolution. This could alternatively be done by transforming the data in original size blocks and averaging the results afterwards. In theory, extremely large amounts of data could be taken if the data acquisition system was configured appropriately. If more than enough resolution for a target was obtained, it would also give the opportunity to reduce the port spacing. The limiting factor in doing this is likely to be the time that is needed to process the data. Having said this, a new motherboard is currently in development for the Acorn RiscPC platform that is capable of taking six Strong Arm processors. With an appropriate multi-tasking C compiler, this might provide a cost effective platform for such a large processing task. Also as this thesis is being written, it is also known that a multi-channel data acquisition board is being developed for the RiscPC. When this board becomes available, it would be prudent to purchase one so that just one machine could be used to handle both data acquisition and processing. This would also offer excellent opportunities for automating experiments.

Apart from the processing aspects of the experiments, there is one

other area that could have been developed further if more time had been available. This basically concerns the positioning of microphones in the anechoic chamber. It would have been nice to find an alternative means of accurately positioning the microphones instead of using the optical rail. The optical rail limited the positions to one dimension, although by using 5 microphones we had the capability for detecting targets in two dimension. In the experiments shown in chapter 6, this problem was overcome by simply turning the loudspeaker on its side and taking a second set of data. A two dimensional frame in which microphones could be automatically positioned, would provide an ideal solution in the future.

7.2 Possible Applications

If in the future the resolution of the system was increased by collecting and processing more data, it would make an ideal test tool for imaging and looking at the spatial coherence properties of loudspeakers. As we have seen, more data means that a narrower bandwidth can be imaged accurately. If enough data was taken, very narrow bandwidths of data could be used for range calculations and a spatial image of the loudspeaker could be built up. In effect, this would mean that different frequencies could be pinpointed to different places within the loudspeaker box. It is envisaged that different qualities of loudspeaker will have very different spatial coherence properties which will no doubt be related to the sound quality. This would be an almost ideal application for the system, because a test loudspeaker could be placed in the anechoic chamber at a position where the maximum resolution is obtained.

Sonar and geophysics are other areas where this system could have applications. In sonar, the microphones could easily be replaced with hydrophones and the speed of sound would need to be adjusted accordingly.

Similarly geophones could be used to detect seismic activity. In both of these cases, it is unknown how well the system would react in a non anechoic environment. Further experiments would also need to be carried out to determine how the system would behave when it sees more than one coherent source. This would probably mean that more than three ports would be required. Care would also need to be taken to fully characterize the differences in resolution that the system sees at different target bearings.

In sum, with a little more effort, the work that has been presented in this thesis offers an ideal opportunity to create a working tool for testing the spatial coherence properties of loudspeakers. Such an application could have big commercial implications and would be likely to offer the audio industry a welcome and new means of characterising and testing loudspeakers.

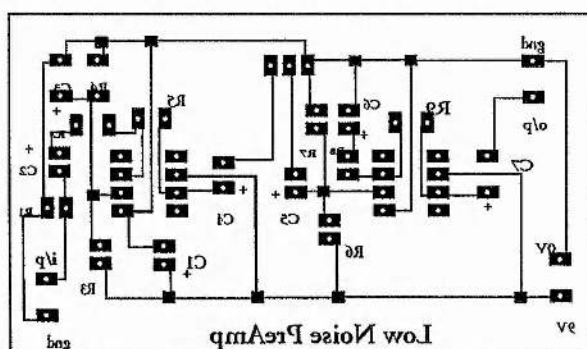
APPENDIX A

Introduction

In this Appendix a few general items are included such as PCBs, technical data, and source code for programs which did not perform digital signal processing. For digital signal processing routines see appendix B.

Pre-Amplifier PCB

This is the PCB that was used for constructing the Pre-Amplifiers described in section 4.4.4 in chapter 4. The schematic and a picture can be found in figure 4.7.



Data Acquisition Program

The following source code was written to take data from the fulcrum dsp board. For more details consult section 4.6 in chapter 4.

```
/* This program is to collect data from a single ADC together
and then output them to a named file */
```

```
#include <spox3801.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cis.h>
#include <gm.h>
```

```
#include <gs.h>
#include <string.h>

/* User defines */

#define ADCNBUFS 25
#define OUTNBUFS 3
#define LOOP 25
#define BUFSIZE 32768

/* smain function */

void smain() {

    ADC_Cgl cgl;
    SA_Array adc_array;
    SS_Stream out_stream,adc_stream;
    int i,loop,status;
    char str[8];
    ADC_Config adc_cfg;
    SS_Attrs ssadc;
    SS_Attrs ssout;
    CIS_Params hst_attrs;
    int chan=0;
    float freq = 52000.0;
    int loopcount = LOOP;
    int chans = 8;
    int bSize;
    char fnamebuf [ 64 ];
    char fname [ 13 ];

    /* Sort out filename */

    printf ( "What is the filename to give the data ? " );
    scanf ( "%s", &fname );

    /* This gives the directory to save the data in */

    strcpy ( fnamebuf, "/file/host:c:\\today\\" );
    strcat ( fnamebuf, fname );

    /* open the devices */

    /* Set up adc config */
```

```

bSize = BUFSIZE;
ssadc.nbufs = ADCNBUFS;
ssadc.align = 0;
ssadc.memseg = SG_DRAM;
ssadc.devattrs = NULL;

/* Open a stream to the ADC */

adc_stream = SS_create ( SS_NULL, -1, bSize, &ssadc );
if (status = SS_open ( adc_stream, "/adc1", SS_READ ) )
    SX_raise ( SX_IO, "FATAL ERROR: can't open /adc1" );

/* Set up file config */

ssout.nbufs = OUTNBUFS;
ssout.align = 0;
ssout.memseg = SG_DRAM;
ssout.devattrs = NULL;
out_stream = SS_create ( SS_NULL, -1, bSize, &ssout );
if (status = SS_open(out_stream,fnamebuf,SS_WRITE)) {
    SX_raise(SX_IO,"FATAL ERROR: can't open file.");
}

/*configure the ADC sample rate*/

SS_ctrl(adc_stream,GET_CONFIG, (Arg) &adc_cfg);

/* freq is set above in the variable declarations */

adc_cfg.rate = freq;
adc_cfg.error_mode = DEV_STOP_ON_ERROR;
adc_cfg.dev.adc.cgl_size = chans;

SS_ctrl(adc_stream,SET_CONFIG, (Arg) &adc_cfg);
SS_ctrl(adc_stream,GET_CONFIG, (Arg) &adc_cfg);

/*create arrays for data minipulation*/
/* An array is need to stream the data in from the ADC, and then
   out to the file*/

adc_array = SA_create(SS_memseg(adc_stream),bSize,NULL);
printf("Copying %d channel(s) of ADC data at %f Hz to output
file\n",chans,adc_cfg.rate);

/*get data, and write to host output file*/

```

```
        for (loop=0;loop<loopcount;loop++) {  
  
            /* get data from ADC and place in array */  
  
                SS_get(adc_stream, adc_array);  
  
            /* Get data from array and place in file */  
  
                SS_put(out_stream, adc_array);  
            }  
  
        /*close devices */  
  
            SS_close(adc_stream);  
            SS_close(out_stream);  
            printf("DMA transfer finished\n");  
  
        /*clean up*/  
  
            SA_free(adc_array);  
            SA_delete(adc_array);  
            SS_delete(out_stream);  
            SS_delete(adc_stream);  
        }
```

Fulcrum Data Acquisition System Specification

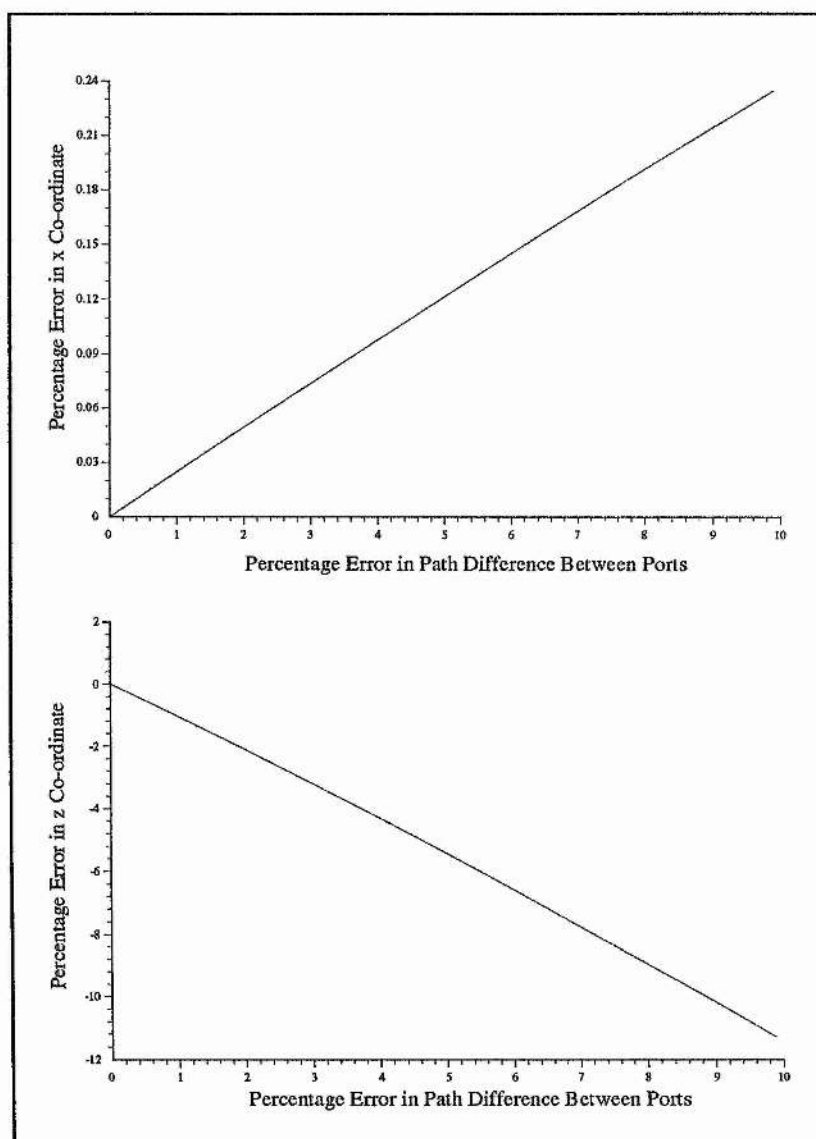
Analog Inputs		
All specifications are typical at 25°C and rated voltage unless otherwise specified.		
	DT3818, DT3818-50	DT3814, DT3814-2
Resolution (bits)	16 (0.0015% FSR)	16 (0.0015% FSR)
Throughput	1-52kHz/channel 416kHz aggregate (8 channels)	1-52kHz/channel 104kHz aggregate (2 channels)
Inputs	8 differential inputs; 8 separate delta sigma converters	2 differential inputs; 2 separate delta sigma converters
Gain	1	1
Range (V)	±10	±10
Digital Anti-Aliasing Filters		
Passband (-3dB)	24kHz	24kHz
Stopband (-80dB)	28kHz	28kHz
Channel-to-Channel Aperture Match (ns)	±5	±5
Channel-to-Channel Phase Match (°)	±0.1	±0.1
Group Delay	18/throughput	18/throughput
Channel-List (entries)	8	2
System Error (% of FSR)	±0.05 max	±0.05 max
Signal/(Noise+Distortion) Ratio (dB) @ f_{in}	89 @ 1kHz in, 48kHz sampling	89 @ 1kHz in, 48kHz sampling
Total Harmonic Distortion (dB) @ f_{in}	-96 @ 1kHz in, 48kHz sampling	-96 @ 1kHz in, 48kHz sampling
Input Impedance	100M Ω /100pF	100M Ω /100pF
CMRR (dB @ 60Hz)	80 min	80 min
Maximum Input Voltage Protection (On/Off)	±25/±15	±25/±15
ESD Protection	To 1.5kV, Mil 38510 class 2	To 1.5kV, Mil 38510 class 2

Analog Outputs* (not available on DT3814)									
	DACs	Resolution (bits)	Throughput (kHz)	Range (V @ ±5mA min)	Total Harmonic Distortion	Recon- struction Filter	Error (% of FSR)	Drift (ppm of FSR/°C)	Group Delay
DT3818, DT3818-50, DT3814-2	2	16 (0.0015% FSR)	1-52kHz/DAC full-scale change	±10	-82dB @ 48kHz w/recon- struction filter	16kHz low-pass (4-pole), software enabled	±0.05	±50	33/ throughput

* All specifications are typical at 25°C and rated voltage, unless otherwise specified.
** Digital anti-aliasing filters automatically operate at the Nyquist frequency (half the sampling rate).

Graphs of Co-ordinate Errors Against Errors in Path Differences Between Ports.

These are examples of the typical error graphs that were referred to in section 6.4.4 in chapter 6.



APPENDIX B

Introduction

The full source code for the digital signal processing routine programs and library functions can be found in this appendix. The Library functions are presented first, followed by the programs that used them. Both of these sections are presented in alphabetical order.

Digital Signal Processing Library (dsp)

This library contains the functions needed for the Fourier transforming and cross correlating data. It contains the following routines:

```
void drealft(double data[], unsigned long n, int isign)  
void xcorr(double data[], double data1[],double out[], unsigned long n)  
void dfour1(double data[], unsigned long nn, int isign)
```

They are now presented in the order as written above. The start of each routine is shown in bold type, making it easier for the reader to find the start and end of each function.

```
#include <math.h>  
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr  
  
void drealft(double data[], unsigned long n, int isign)  
{  
    void dfour1(double data[], unsigned long nn, int isign);  
    unsigned long i,i1,i2,i3,i4,np3;  
    double c1=0.5,c2,h1r,h1i,h2r,h2i;  
    double wr,wi,wpr,wpi,wtemp,theta;  
  
    theta=3.141592653589793/(double) (n>>1);  
    if (isign == 1) {  
        c2 = -0.5;
```

```

    dfour1(data,n>>1,1);
} else {
    c2=0.5;
    theta = -theta;
}
wtemp=sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0+wpr;
wi=wpi;
np3=n+3;
for (i=2;i<=(n>>2);i++) {
    i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
    h1r=c1*(data[i1]+data[i3]);
    h1i=c1*(data[i2]-data[i4]);
    h2r = -c2*(data[i2]+data[i4]);
    h2i=c2*(data[i1]-data[i3]);
    data[i1]=h1r+wr*h2r-wi*h2i;
    data[i2]=h1i+wr*h2i+wi*h2r;
    data[i3]=h1r-wr*h2r+wi*h2i;
    data[i4] = -h1i+wr*h2i+wi*h2r;
    wr=(wtemp=wr)*wpr-wi*wpi+wr;
    wi=wi*wpr+wtemp*wpi+wi;
}
if (isign == 1) {
    data[1] = (h1r=data[1])+data[2];
    data[2] = h1r-data[2];
} else {
    data[1]=c1*((h1r=data[1])+data[2]);
    data[2]=c1*(h1r-data[2]);
    dfour1(data,n>>1,-1);
}
}

void xcorr(double data[], double data1[],double out[], unsigned long n)
{
    unsigned long i;
    double extra, temp1, temp2;
    i=0;
    extra=((data[1]*data1[1])/(double)((double)(n)/2));
    temp1=data[1];
    temp2=data1[1];
    data[1]=0;
    data1[1]=0;
    do {
        out[i]=(((data[i]*data1[i])+(data[i+1]*data1[i+1]))/(double)((double)(n)/2));
        out[i+1]=(((data[i+1]*data1[i])-(data[i]*data1[i+1]))/(double)((double)(n)/2));
        i+=2;
    } while (i<n);
    out[1]=extra;
    data[1]=temp1;
    data1[1]=temp2;
}

```

```

void dfour1(double data[], unsigned long nn, int isign)
{
    unsigned long n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;
    double tempr,tempi;

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {
        if (j > i) {
            SWAP(data[j],data[i]);
            SWAP(data[j+1],data[i+1]);
        }
        m=n >> 1;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
    mmax=2;
    while (n > mmax) {
        istep=mmax << 1;
        theta=isign*(6.28318530717959/mmax);
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;
        for (m=1;m<mmax;m+=2) {
            for (i=m;i<=n;i+=istep) {
                j=i+mmax;
                tempr=wr*data[j]-wi*data[j+1];
                tempi=wr*data[j+1]+wi*data[j];
                data[j]=data[i]-tempr;
                data[j+1]=data[i+1]-tempi;
                data[i] += tempr;
                data[i+1] += tempi;
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
}
#ifdef SWAP

```

File Library (Filz)

This library provides the functions needed to read input data from the data acquisition system and to write output results to disc. The following routines can be found:

```
void Load_Data(int data[])  
void Ram_write(double data[],const char fname[],int start_point, int end_point, double xinc)  
void Ram_write2(double data[],double data2[],const char fname[],int start_point, int  
end_point, double xinc)
```

They are now presented in the order as written above. The start of each routine is shown in bold type, making it easier for the reader to find the start and end of each function.

```
/* filz.c */  
#include <stdio.h>  
#include <string.h>  
#include <kernel.h>  
#include <swis.h>  
#include <wimp.h>  
  
_kernel_swi_regs rin,rout;  
char fname[64]="ADFS::HardDisc4.$.test.test1\0";  
  
FILE *fp;  
  
void Load_Data(int data[])  
{  
    rin.r[0]=16;  
    rin.r[1]=(int)fname;  
    rin.r[2]=(int)data;  
    rin.r[3]=0;  
  
    _kernel_swi(OS_File,&rin,&rout);  
}  
  
void Ram_write(double data[],const char fname[],int start_point, int end_point, double xinc)  
{  
    char path[64]="ram:\0";
```

```
int i;
double temp;
i=start_point;
strcat(path,fname);
fp=fopen(path,"wt");
do {
    temp=((xinc)*(double)(i));
    fprintf(fp,"%lf, %lf\n",temp, data[i]);
    i++;
} while (i<end_point);
fclose(fp);
}
```

```
void Ram_write2(double data[],double data2[],const char fname[],int start_point, int
end_point, double xinc)
{
    char path[64]="ram:\0";
    int i;
    double temp;
    i=start_point;
    strcat(path,fname);
    fp=fopen(path,"wt");
    do {
        temp=((xinc)*(double)(i));
        fprintf(fp,"%lf, %lf, %lf, %lf\n",temp, data[i], temp, data2[i]);
        i++;
    } while (i<end_point);
    fclose(fp);
}
```

System Utility Library (nrutil)

This library contains the routines needed to allocate and deallocate arrays of memory and report any errors generated. The start of each routine is shown in bold type. For more information see chapter 5, section 5.5.

```
/* nrutils */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(char error_text[])
/* Standard error handler */
{
    fprintf(stderr, "Run-time error...\n");
    fprintf(stderr, "%s\n", error_text);
    fprintf(stderr, "...now exiting to system...\n");
    exit(1);
}

float *vector(long nl, long nh)
/* allocate a float vector with subscript range v[nl..nh] */
{
    float *v;
    v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl+NR_END;
}

short *svector(long nl, long nh)
/* allocate an short vector with subscript range v[nl..nh] */
{
    short *v;
    v=(short *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(short)));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl+NR_END;
}

int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{
    int *v;
    v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
```



```

if (!v) perror("allocation failure in ivector()");
return v-nl+NR_END;
}

```

```

unsigned char *cvector(long nl, long nh)
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
    unsigned char *v;
    v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned char)));
    if (!v) perror("allocation failure in cvector()");
    return v-nl+NR_END;
}

```

```

unsigned long *lvector(long nl, long nh)
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
    unsigned long *v;
    v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
    if (!v) perror("allocation failure in lvector()");
    return v-nl+NR_END;
}

```

```

double *dvector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
    double *v;
    v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
    if (!v) perror("allocation failure in dvector()");
    return v-nl+NR_END;
}

```

```

void free_vector(float *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

```

```

void free_svector(short *v, long nl, long nh)
/* free a short vector allocated with ivector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

```

```

void free_ivector(int *v, long nl, long nh)
/* free an int vector allocated with ivector() */
{
    free((FREE_ARG) (v+nl-NR_END));
}

```

```
void free_cvector(unsigned char *v, long nl, long nh)  
/* free an unsigned char vector allocated with cvector() */  
{  
    free((FREE_ARG) (v+nl-NR_END));  
}
```

```
void free_lvector(unsigned long *v, long nl, long nh)  
/* free an unsigned long vector allocated with lvector() */  
{  
    free((FREE_ARG) (v+nl-NR_END));  
}
```

```
void free_dvector(double *v, long nl, long nh)  
/* free a double vector allocated with dvector() */  
{  
    free((FREE_ARG) (v+nl-NR_END));  
}
```

Range Library (Range)

This library contains the function which was used to find the co-ordinates of a target from the path difference between channels. For more details see chapter 5, section 5.6.2.

```
/* Range.c */

#include <stdio.h>
#include <math.h>

void Range(double m, double n)
{
    double x,y,a,t1,t2,t3;
    a=40.00;
    t1=m*((a*a)-(n*n));
    t2=n*((m*m)-(a*a));
    t3=2*a*(n-m);
    x=(t1-t2)/t3;
    t1=(2*a*x)+(a*a)-(n*n);
    t2=t1/(2.0*n);
    t3=(t2*t2)-(x*x);
    y=sqrt(t3);
    printf(" coords are %lf,%lf",x,y);
}
```

Statistics Library (stats)

This library contains the routines needed to perform linear regression and a mean average. For more details see chapter 5, section 5.6.1

```
/* stats.c */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double LRyx(double data[], unsigned long n)
{
    double xb,yb,sxy,sxx,xb2;
    double temp, temp1, temp2,grad, a, b,inc;
    int i;

    i=0; xb=0; yb=0; sxy=0; sxx=0; xb2=0;
    inc=12000.00/1024.00;

    do {
        xb+=((double)(i)*inc/(double)(n));
        yb+=(data[i]/(double)(n));
        sxy+=(double)(i)*inc*(data[i]);
        sxx+=(double)(i)*(double)(i)*inc*inc;
        i++;
    } while (i<n);
    xb2=(xb*xb);
    /*
    printf("xb= %lf\n",xb);
    printf("yb= %lf\n",yb);
    printf("sxy= %lf\n",sxy);
    printf("sxx= %lf\n",sxx);
    printf("xb2= %lf\n",xb2);
    */
    temp1=sxy-((double)(n)*xb*yb);
    temp2=sxx-((double)(n)*xb2);
    temp=(temp1/temp2);
    //printf("temp1= %lf\n",temp1);
    //printf("temp1= %lf\n",temp2);

    a=(temp*(0-xb))+yb;
    b=(temp*((double)(n)-1.0-xb))+yb;

    grad=(double)(b-a)/((double)(n)-1);
    printf("grad= %lf\n",grad);
    return grad;
}
```

```
double MeanAv(double data[], int start, int finnish)
{
    int no_values,i;
    double mean, temp;
    no_values=finnish-start+1;
    temp=0;
    i=start;
    do {
        temp+=(data[i])/((double)(no_values));
        i++;
    } while (i<=finnish);
    return mean;
}
```

DSP Utility Library (utilz)

This library contains the routines discussed in Chapter 5, section 5.4. The routines were used for preparing data for input to the digital signal processing routines and for presenting the results afterwards.

```
/* utilz.c */

#include <math.h>
#include <stdio.h>

void bartlett(double data[], unsigned long n)
{
    unsigned long i;
    double temp, temp1, temp2;
    i=0;
    do {
        temp=(double)((i-(n/2))/(n/2));
        temp2=1-fabs(temp);
        temp1=data[i];
        data[i]=temp2*temp1;
        i++;
    } while (i<n);
}

void rmdc(double data[], unsigned long n)
{
    unsigned long i;
    double temp, temp1;
    i=0;
    temp=0;
    do {
        temp+=data[i];
        i++;
    } while (i<n);
    temp1=(temp/n);
    i=0;
    do {
        temp=data[i];
        data[i]=(temp-temp1);
        i++;
    } while (i<n);
}

void smoother(double data[], unsigned long n, unsigned long no_points)
{
    unsigned long i,j,k;
    double temp;
```



```
i=0;
j=0;
k=0;
do {
    j=0;
    temp=0;
    do {
        temp+=data[i];
        i++;
        j++;
    } while (j<no_points);
    data[k]=(temp/no_points);
    k++;
} while (k<n/no_points);
}
```

void db(double data[], unsigned long n)

```
{
    double max_val, temp;
    int i;
    i=0;
    max_val=0.000;
    do {
        if (max_val<data[i]) max_val=data[i];
        i++;
    } while (i<n);
    printf("max val is....%lf\n",max_val);
    i=0;
    do{
        //temp=20*log10(data[i]/max_val);

        temp=20*log10(data[i]/32768);

        data[i]=temp;
        i++;
    } while (i<n);
}
```

void phase(double data[], unsigned long n)

```
{
    unsigned long i,j;
    double temp;
    i=0;
    j=0;
    do {
        temp = atan2(data[i+1],data[i]);
        data[j]=temp;
        i+=2;
        j++;
    } while (i<n);
}
```

```
void diff(double data[], unsigned long n)
```

```
{
    unsigned long i,j;
    double temp,pi,pi2;
    pi=3.1415927; pi2=2.0*pi;
    i=0;
    j=0;
    do {
        temp=(data[i+1]-data[j]);
        if (i>200)
        {
            if (temp>pi) temp-=pi2;
            if (temp<(-pi)) temp+=pi2;
        }
        data[j]=temp;
        i+=1;
        j++;
    } while (i<(n-1));
}
```

```
void stitch(double data[], unsigned long n, unsigned long skip)
```

```
{
    int i,j;
    double temp, temp1, temp2, temp3, temp4, temp5, temp6, temp7, temp8, mean, pi2;
    pi2=6.283185307;
    i=(int)skip;
    temp1=data[skip];
    temp2=data[skip-1];
    temp3=data[skip-2];
    temp4=data[skip-3];
    temp5=data[skip-4];
    temp6=data[skip-5];
    temp7=data[skip-6];
    temp8=data[skip-7];
    mean=((temp1+temp2+temp3+temp4+temp5+temp6+temp7+temp8)/8.00);
    do {
        j=0;
        do {
            temp=(data[i+1]-mean);
            if (temp>=3.14) data[i+1]-=pi2;
            if (temp<(-3.14)) data[i+1]+=pi2;
            j++;
        } while (j<8);
        temp8=temp7;
        temp7=temp6;
        temp6=temp5;
        temp5=temp4;
        temp4=temp3;
        temp3=temp2;
        temp2=temp1;
        temp1=data[i+1];
        mean=((temp1+temp2+temp3+temp4+temp5+temp6+temp7+temp8)/8.00);
        i++;
    }
```

```
    } while (i<(n-1));  
}
```

void stitch1(double data[], unsigned long n, unsigned long skip)

```
{  
    int i,j,k;  
    double mean, temp,pi2;  
    pi2=6.283185307;  
    i=(int)skip;  
    mean=0;  
    k=0;  
    do {  
        mean+=(data[i])/(20.00);  
        i++;  
        k++;  
    } while (k<20);  
  
    do {  
        j=0;  
        do {  
            temp=(data[i]-mean);  
            if (temp>=3.14) data[i]-=pi2;  
            if (temp<(-3.14)) data[i]+=pi2;  
            j++;  
        } while (j<8);  
        j=i;  
        k=0;  
        mean=0;  
        do {  
            mean+=(data[j])/(20.00);  
            j--;  
            k++;  
        } while (k<20);  
        i++;  
    } while (i<n);  
}
```

void ConReg(double data[], unsigned long n, unsigned long miss)

```
{  
    int i;  
    double temp;  
    i=(int)(miss);  
    do {  
        temp=data[i]/((double)(i));  
        data[i]=temp;  
        i++;  
    } while (i<n);  
}
```

void power(double data[], unsigned long n)

```
{
    unsigned long i,j;
    double temp;
    i=0;
    j=0;
    do {
        temp = sqrt(((data[i]*data[i])+(data[i+1]*data[i+1])));
        data[j]=temp;
        i+=2;
        j++;
    } while (i<n);
}
```

Programs Using the Library Routines

The following pages contain the code for programs that are made referenced to in this thesis. They all rely on the use of the library functions that were presented at the beginning of this appendix. To see which library functions are required for each program, one should look at the start of the code for the "#include" statements. The programs are now presented in alphabetical order below.

4 Channel Spectrum Analyser

A 4 channel spectrum analyser to produce the spectra of the first 4 channels of a data set taken with the data acquisition system. It would have been a trivial task to extend the program to cater for more channels.

```
/* 4 Channel Spectrum Analyser */

#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#include "dsp.h"
#include "utilz.h"
#include "filz.h"

double *ch1,*ch2,*ch3,*ch4,*out;
double temp;
unsigned long no_samples;
int *nums;
int i,j;

int main(void)
{
    nums=ivector(0,819199);

    no_samples=65536;
    Load_Data(nums);

    /* put data into array for each channel */

    ch1=dvector(0,65535);
    ch2=dvector(0,65535);
```

```
ch3=dvector(0,65535);
ch4=dvector(0,65535);
out=dvector(0,65535);
i=0;
j=0;
do {
    ch1[j]=(((double)(nums[j])/32768.00));
    ch2[j]=(((double)(nums[i+1])/32768.00));
    ch3[j]=(((double)(nums[i+2])/32768.00));
    ch4[j]=(((double)(nums[i+3])/32768.00));
    i+=8;
    j++;
} while (j<65536);

/* do the nasty dsp */

rmdc(ch1,no_samples);
rmdc(ch2,no_samples);
rmdc(ch3,no_samples);
rmdc(ch4,no_samples);
drealft(ch1-1,no_samples,1);
drealft(ch2-1,no_samples,1);
drealft(ch3-1,no_samples,1);
drealft(ch4-1,no_samples,1);
power(ch1,no_samples);
power(ch2,no_samples);
power(ch3,no_samples);
power(ch4,no_samples);
smoother(ch1,no_samples,64);
smoother(ch2,no_samples,64);
smoother(ch3,no_samples,64);
smoother(ch4,no_samples,64);

db(ch1,512);
db(ch2,512);
db(ch3,512);
db(ch4,512);

Ram_write(ch1,"ch1",0,32768/64,26002.83984/1024);
Ram_write(ch2,"ch2",0,32768/64,26002.83984/1024);
Ram_write(ch3,"ch3",0,32768/64,26002.83984/1024);
Ram_write(ch4,"ch4",0,32768/64,26002.83984/1024);

free_ivector(nums,0,819199);
free_dvector(ch1,0,65535);
free_dvector(ch2,0,65535);
free_dvector(ch3,0,65535);
free_dvector(ch4,0,65535);
free_dvector(out,0,65535);

return 0;
}
```

Decibel Scale Calibration Program (dbRef)

```
/* Zero db calibration program */

#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#include "dsp.h"
#include "utilz.h"
#include "filz.h"

double *ch1,*out;
double temp,PI2;
int i,j;

int main(void)
{
    ch1=dvector(0,65535);
    out=dvector(0,65535);
    PI2=6.283185307;
    j=0;
    do {
        i=0;
        do {
            temp=((PI2*(double)(i))/32.000);
            ch1[j]=4.000*sin(temp);
            i++;
            j++;
        } while (i<32);
    } while (j<65535);

    /* do the nasty dsp */

    drealf(ch1-1,16384,1);
    power(ch1,16384);
    db(ch1,8192);

    Ram_write(ch1,"ch1",0,16384,1);

    free_dvector(ch1,0,65535);
    free_dvector(out,0,65535);
    return 0;
}
```


Digital Signal Processing Main Program (dspProj)

The following program is one of many that was used for performing digital signal processing routines on experimental data. This program has been included as an example to show how easily such a program can be written using the library functions.

```
/* dspProj.c */

#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#include "dsp.h"
#include "utilz.h"
#include "filz.h"
#include "stats.h"
#include "range.h"

double *ch1,*ch2,*ch3,*ch4,*out1,*out2;
double temp, grad1, grad2, dist1, dist2;
double grad3, grad4, dist3, dist4;
unsigned long no_samples;
int *nums;
int i,j;

int main(void)
{
    nums=ivector(0,819199);
    Load_Data(nums);
    no_samples=65536;

    /* put data into array for each channel */

    ch1=dvector(0,65535);
    ch2=dvector(0,65535);
    ch3=dvector(0,65535);
    ch4=dvector(0,65535);
    out1=dvector(0,65535);
    out2=dvector(0,65535);
    i=0;
    j=0;
    do {
        ch1[j]=(((double)(nums[i+2])/32768.00));
        ch2[j]=(((double)(nums[i+3])/32768.00));
        ch3[j]=(((double)(nums[i+4])/32768.00));
        ch4[j]=(((double)(nums[i+3])/32768.00));
        i+=8;
        j++;
    }
```

```

} while (j<65536);

/* do the nasty dsp */

rmdc(ch1,no_samples);
rmdc(ch2,no_samples);
rmdc(ch3,no_samples);
rmdc(ch4,no_samples);
drealft(ch1-1,no_samples,1);
drealft(ch2-1,no_samples,1);
drealft(ch3-1,no_samples,1);
drealft(ch4-1,no_samples,1);
xcorr(ch2,ch3,out1,no_samples);
phase(out1,no_samples);
stitch1(out1,32768,30);
grad1=L.Ryx(out1+100,28000);
printf(" grad1=%lf\n",grad1);
dist1=grad1*330/(2.0*3.14159265);
printf(" dist1=%lf\n",dist1);
grad3=L.Ryx(out1+10900,20000);
printf(" grad3=%lf\n",grad3);
dist3=grad3*330/(2.0*3.14159265);
printf(" dist3=%lf\n",dist3);
xcorr(ch1,ch2,out2,no_samples);
phase(out2,no_samples);
stitch1(out2,32768,30);
grad2=L.Ryx(out2+100,28000);
printf(" grad2=%lf\n",grad2);
dist2=grad2*330/(2.0*3.14159265);
printf(" dist2=%lf\n",dist2);
grad4=L.Ryx(out2+10900,20000);
printf(" grad4=%lf\n",grad4);
dist4=grad4*330/(2.0*3.14159265);
printf(" dist4=%lf\n",dist4);
Range(dist1*100.00,dist2*100.00);
Range(dist3*100.00,dist4*100.00);
smoother(out1,32768,32);
Ram_write(out1,"2x3b",0,1024,12000.00/1024.00);
smoother(out2,32768,32);
Ram_write(out2,"1x2b",0,1024,12000.00/1024.00);

free_ivector(nums,0,819199);
free_dvector(ch1,0,65535);
free_dvector(ch2,0,65535);
free_dvector(ch3,0,65535);
free_dvector(ch4,0,65535);
free_dvector(out1,0,65535);
free_dvector(out2,0,65535);

return 0;
}

```

Errors

This program was used for creating the error data shown in chapter 3 in section 3.5.

```
/* errors.c */

#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#include "filz.h"
#include "range.h"

int main(void)
{
    double *errorx, *error;
    double s1x,s1y,s1z,s2x,s2y,s2z,p1x,p1y,p1z,p2x,p2y,p2z,p3x,p3y,p3z;
    double d11, d12, d21, d22, d31, d32;
    double n1,n2,m1,m2;
    double x1,y1,x2,y2;
    double xref, yref, errinc, final, port_space;
    int i;
    errorx=dvector(0,1000);
    error=dvector(0,1000);

    /* geometry variables */

    port_space=0.20;

    s1x=-0.05;
    s1y=0.000;
    s1z=1.700;
    s2x=0.100;
    s2y=0.000;
    s2z=1.700;
    p1x=(0.0-port_space);
    p1y=0.000;
    p1z=0.000;
    p2x=0.000;
    p2y=0.000;
    p2z=0.000;
    p3x=port_space;
    p3y=0.000;
    p3z=0.000;

    d11=sqrt( pow((s1x-p1x),2) + pow((s1y-p1y),2) + pow((s1z-p1z),2));
    d12=sqrt( pow((s2x-p1x),2) + pow((s2y-p1y),2) + pow((s2z-p1z),2));
    d21=sqrt( pow((s1x-p2x),2) + pow((s1y-p2y),2) + pow((s1z-p2z),2));
    d22=sqrt( pow((s2x-p2x),2) + pow((s2y-p2y),2) + pow((s2z-p2z),2));
```

```

d31=sqrt( pow((s1x-p3x),2) + pow((s1y-p3y),2) + pow((s1z-p3z),2));
d32=sqrt( pow((s2x-p3x),2) + pow((s2y-p3y),2) + pow((s2z-p3z),2));
/* calculate m and n */
n1=(d11-d21);
n2=(d12-d22);
m1=(d21-d31);
m2=(d22-d32);
printf("n1=%lf n2=%lf m1=%lf m2=%lf", n1, n2, m1, m2);
x1=RangeX(m1,n1,port_space);
y1=RangeY(m1,n1,port_space);
x2=RangeX(m2,n2,port_space);
y2=RangeY(m2,n2,port_space);
xref=x2;
yref=y2;
errorx[0]=0;
errory[0]=0;
printf("x1=%lf,y1=%lf",x1,y1);
printf("x2=%lf,y2=%lf",x2,y2);
/* choose port p3x for errors */
final=(p3x*1.10);
errinc=(0.001*p3x);
p3x+=errinc;
//p1x-=errinc;
i=1;
do {
    d11=sqrt( pow((s1x-p1x),2) + pow((s1y-p1y),2) + pow((s1z-p1z),2));
    d12=sqrt( pow((s2x-p1x),2) + pow((s2y-p1y),2) + pow((s2z-p1z),2));
    d21=sqrt( pow((s1x-p2x),2) + pow((s1y-p2y),2) + pow((s1z-p2z),2));
    d22=sqrt( pow((s2x-p2x),2) + pow((s2y-p2y),2) + pow((s2z-p2z),2));
    d31=sqrt( pow((s1x-p3x),2) + pow((s1y-p3y),2) + pow((s1z-p3z),2));
    d32=sqrt( pow((s2x-p3x),2) + pow((s2y-p3y),2) + pow((s2z-p3z),2));
    n1=(d11-d21);
    n2=(d12-d22);
    m1=(d21-d31);
    m2=(d22-d32);
    x1=RangeX(m1,n1,port_space);
    y1=RangeY(m1,n1,port_space);
    x2=RangeX(m2,n2,port_space);
    y2=RangeY(m2,n2,port_space);
    errorx[i]=((xref-x2)/xref)*100;
    errory[i]=((yref-y2)/yref)*100;
    i++;
    p3x+=errinc;
    //p1x-=errinc;
} while (p3x<final);

Ram_write(errorx,"Xerr",0,100,0.1);
Ram_write(errory,"Yerr",0,100,0.1);

return 0;
}

```

Modelling Program (Model)

```
/*model.c*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "nrutil.h"
#include "filz.h"

int main(void)
{
    double *phase,*ch1,*ch2,*ch3;
    int i, j, seed, no_samples;
    double rnd, temp, pi2,sample_rate,delta_t, freq_int, cross_freq;
    double s1x,s1y,s1z,s2x,s2y,s2z,p1x,p1y,p1z,p2x,p2y,p2z,p3x,p3y,p3z;
    double d11, d12, d21, d22, d31, d32;
    pi2=6.283185;
    phase=dvector(0,65535);
    ch1=dvector(0,65535);
    ch2=dvector(0,65535);
    ch3=dvector(0,65535);

    /* setup rand phase array */

    seed = time(NULL);
    srand(seed);
    printf("RAND_MAX = %d\n", RAND_MAX);
    i=0;
    do {
        rnd = pi2*((double)(rand()/(double)(RAND_MAX)));
        phase[i]=rnd;
        i++;
    } while (i<65536);

    /* setup input variables */

    sample_rate=24000.00000;
    delta_t=1.0000/sample_rate;
    freq_int=4;
    no_samples=65536;
    cross_freq=3200.000;

    /* geometry variables */

    s1x=-0.05;
    s1y=0.000;
    s1z=1.700;
    s2x=0.100;
    s2y=0.000;
```

```

s2z=1.700;
p1x=-0.20;
p1y=0.000;
p1z=0.000;
p2x=0.000;
p2y=0.000;
p2z=0.000;
p3x=0.200;
p3y=0.000;
p3z=0.000;

d11=sqrt( pow((s1x-p1x),2) + pow((s1y-p1y),2) + pow((s1z-p1z),2));
d12=sqrt( pow((s2x-p1x),2) + pow((s2y-p1y),2) + pow((s2z-p1z),2));
d21=sqrt( pow((s1x-p2x),2) + pow((s1y-p2y),2) + pow((s1z-p2z),2));
d22=sqrt( pow((s2x-p2x),2) + pow((s2y-p2y),2) + pow((s2z-p2z),2));
d31=sqrt( pow((s1x-p3x),2) + pow((s1y-p3y),2) + pow((s1z-p3z),2));
d32=sqrt( pow((s2x-p3x),2) + pow((s2y-p3y),2) + pow((s2z-p3z),2));

printf("d11=%lf\nd12=%lf\nd21=%lf\nd22=%lf\nd31=%lf\nd32=%lf\n",d11,d12,d21,d22,d31,d32);

/* do channel 1 */

j=0;
do {
    ch1[j]=0.00;
    i=0;
    do {
        temp=(pi2*((double)(i)*freq_int)*(((double)(j)*delta_t)-(d11/330.00)))+phase[i];
        ch1[j]+=cos(temp);
        i++;
    } while (((double)(i)*freq_int)<cross_freq);
    do {
        temp=(pi2*((double)(i)*freq_int)*(((double)(j)*delta_t)-(d12/330.00)))+phase[i];
        ch1[j]+=cos(temp);
        i++;
    } while (((double)(i)*freq_int)<(sample_rate/2));

    j++;
    printf("Calculating cell no %d\n",j);
} while (j<no_samples);

Ram_write(ch1,"ch1",0.65536,1);

/* do channel 2 */

j=0;
do {
    ch2[j]=0.00;
    i=0;
    do {
        temp=(pi2*((double)(i)*freq_int)*(((double)(j)*delta_t)-(d21/330.00)))+phase[i];
        ch2[j]+=cos(temp);

```

```
i++;
} while (((double)(i)*freq_int)<cross_freq);
do {
    temp=(pi2*((double)(i)*freq_int)*(((double)(j)*delta_t)-(d22/330.00)))+phase[i];
    ch2[j]+=cos(temp);
    i++;
} while (((double)(i)*freq_int)<(sample_rate/2));

j++;
printf("Calculating cell no %d\n",j);
} while (j<no_samples);

Ram_write(ch2,"ch2",0,65536,1);

/* do channel 3 */

j=0;
do {
    ch3[j]=0.00;
    i=0;
    do {
        temp=(pi2*((double)(i)*freq_int)*(((double)(j)*delta_t)-(d31/330.00)))+phase[i];
        ch3[j]+=cos(temp);
        i++;
    } while (((double)(i)*freq_int)<cross_freq);
    do {
        temp=(pi2*((double)(i)*freq_int)*(((double)(j)*delta_t)-(d32/330.00)))+phase[i];
        ch3[j]+=cos(temp);
        i++;
    } while (((double)(i)*freq_int)<(sample_rate/2));

    j++;
    printf("Calculating cell no %d\n",j);
} while (j<no_samples);

Ram_write(ch3,"ch3",0,65536,1);

free_dvector(phase,0,65535);
free_dvector(ch1,0,65535);
free_dvector(ch2,0,65535);
free_dvector(ch3,0,65535);

return 0;
}
```


Model Data Processing Program (Promodel)

```
/* Model process routine */

#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#include "dsp.h"
#include "utilz.h"
#include "filz.h"
#include "stats.h"
#include "range.h"

FILE *ifp;

double *ch1,*ch2,*ch3,*out1,*out2;

int main(void)
{
    int i;
    double temp, temp1;
    double grad1, grad2, dist1, dist2;
    double grad3, grad4, dist3, dist4;
    unsigned long no_samples;
    ch1=dvector(0,65535);
    ch2=dvector(0,65535);
    ch3=dvector(0,65535);
    out1=dvector(0,65535);
    out2=dvector(0,65535);
    no_samples=65536;

    /* Load The Data */

    ifp=fopen("ram:ch1\0","rt");
    for(i=0;i<65536;i++)
    {
        fscanf(ifp,"%lf,%lf",&temp,&temp1);
        ch1[i]=temp1;
    }
    fclose(ifp);

    ifp=fopen("ram:ch2\0","rt");
    for(i=0;i<65536;i++)
    {
        fscanf(ifp,"%lf,%lf",&temp,&temp1);
        ch2[i]=temp1;
    }
    fclose(ifp);

    ifp=fopen("ram:ch3\0","rt");
    for(i=0;i<65536;i++)
```

```

{
    fscanf(ifp,"%lf,%lf",&temp,&temp1);
    ch3[i]=temp1;
}
fclose(ifp);

/* do the nasty DSP */

rmdc(ch1,no_samples);
rmdc(ch2,no_samples);
rmdc(ch3,no_samples);

drealft(ch1-1,no_samples,1);
drealft(ch2-1,no_samples,1);
drealft(ch3-1,no_samples,1);

xcorr(ch2,ch3,out1,no_samples);
phase(out1,no_samples);
stitch1(out1,32768,30);
//smoother(out1,32768,64);
Ram_write(out1,"2x3b",0,32768,1);
grad1=L_Ryx(out1+20,8000);
printf("grad1=%lf\n",grad1);
dist1=grad1*330/(2.0*3.14159265);
printf("dist1=%lf\n",dist1);
grad3=L_Ryx(out1+10900,20000);
printf("grad3=%lf\n",grad3);
dist3=grad3*330/(2.0*3.14159265);
printf("dist3=%lf\n",dist3);
xcorr(ch1,ch2,out2,no_samples);
phase(out2,no_samples);
stitch1(out2,32768,30);
//smoother(out2,32768,64);
Ram_write(out2,"1x2b",0,32768,1);
grad2=L_Ryx(out2+20,8000);
printf("grad2=%lf\n",grad2);
dist2=grad2*330/(2.0*3.14159265);
printf("dist2=%lf\n",dist2);
grad4=L_Ryx(out2+10900,20000);
printf("grad4=%lf\n",grad4);
dist4=grad4*330/(2.0*3.14159265);
printf("dist4=%lf\n",dist4);

Range(dist1*100.00,dist2*100.00);
Range(dist3*100.00,dist4*100.00);

free_dvector(ch1,0,65535);
free_dvector(ch2,0,65535);
free_dvector(ch3,0,65535);
free_dvector(out1,0,65535);
free_dvector(out2,0,65535);
return 0;
}

```

Stanford Spectrum Analyser Emulation Routines

Here two pieces of code are included. The first called "Stan_Av" is a library function, although it actually calls a number of other library functions. The second is a program which emulates a stanford specrum analyser and is called "StanEmul". This program makes use of the "Stan_Av" library.

Stan_Av

```
/* Stan_Av.c */

#include <stdio.h>
#include "dsp.h"
#include "nrutil.h"
#include "utilz.h"

void Stan_Av(double data[], unsigned long fft_size)
{
    int i,j,k;
    i=0;
    do {
        j=i-1;
        drealfit(data+j,fft_size,1);
        power(data+i,fft_size);
        i+=(int)fft_size;
        k++;
    } while (i<102400);
    printf("Averaged %i times using %i point fft's\n",k,fft_size);

    /* now do averaging */

    i=fft_size;
    do {
        j=0;
        do {
            data[j]+=data[i];
            i++;
            j++;
        } while (j<fft_size/2);
        i+=(fft_size/2);
    } while (i<102400);
}
```

StanEmul

```
/* stanford emulation program */

#include <stdio.h>
#include <math.h>
#include "nrutil.h"
#include "dsp.h"
#include "utilz.h"
#include "filz.h"
#include "Stan_Av.h"
double *ch1,*ch2,*ch3,*ch4;
double temp;
unsigned long no_samples;
int *nums;
int i,j;

int main(void)
{
    nums=ivector(0,819199);
    no_samples=65536;
    Load_Data(nums);

    /* put data into array for each channel */

    ch1=dvector(0,102399);
    ch2=dvector(0,102399);
    ch3=dvector(0,102399);
    ch4=dvector(0,102399);

    i=0;
    j=0;
    do {
        ch1[j]=(((double)(nums[i])/32768.00));
        ch2[j]=(((double)(nums[i+1])/32768.00));
        ch3[j]=(((double)(nums[i+2])/32768.00));
        ch4[j]=(((double)(nums[i+3])/32768.00));
        i+=8;
        j++;
    } while (j<102400);

    Stan_Av(ch1,1024);
    db(ch1,512);
    Ram_write(ch1,"stan1",0,512,1);

    free_ivector(nums,0,819199);
    free_dvector(ch1,0,102399);
    free_dvector(ch2,0,102399);
    free_dvector(ch3,0,102399);
    free_dvector(ch4,0,102399);

    return 0;
}
```